universität
wien

DIPLOMARBEIT

Titel der Diplomarbeit

Modellierung und Implementation eines Vorlesungsplaners

angestrebter akademischer Grad

Magister der Naturwissenschaften (Mag. rer. nat.)

Verfasser:	Harald Schilly
Matrikel-Nummer:	9900479
Studienrichtung:	Mathematik
Betreuer:	Ao. Univ.-Prof. Dipl.-Ing. Dr. Hermann Schichl

Wien, am 21. 6. 2007

Inhaltsverzeichnis

0	**Einleitung**	**1**
1	**Hintergrund**	**3**
1.1	Problembehandlung	4
1.2	Suchalgorithmen	6
1.2.1	Komplexität	6
1.2.2	Simulated Annealing	6
1.2.3	Tabu Search	8
1.2.3.1	Tabukriterien	10
1.2.3.2	Aspirationskriterien	11
1.2.3.3	Beispiel	11
1.2.3.4	Verbesserungen	12
1.2.4	Constraint Propagation	12
1.2.4.1	Knotenkonsistenz	13
1.2.4.2	Kantenkonsistenz	13
1.2.4.2.1	Beispiel	15
1.2.4.3	Suchmethoden	16
1.2.4.3.1	backtracking	16
1.2.4.3.2	forward-checking	17
1.2.4.3.3	look-ahead	18
1.2.4.4	Hilfsmittel	20
1.2.4.4.1	cycle-cutset	20
1.2.4.4.2	MACE	21
1.2.4.4.3	zyklische Untergraphen	21
1.2.4.5	Nachbemerkung	21
1.2.4.5.1	Überbestimmte CSPs	22
1.2.5	Kombinierte Algorithmen	22
1.2.6	Reihung der Variablen und Wertzuweisung	23
1.2.6.1	Reihung der Variablenzuweisung	23
1.2.6.2	Reihung der Wertzuweisung	24
2	**Problembeschreibung**	**25**
2.1	Allgemein	25
2.2	Modell und Bedingungen	25
2.2.1	Randbedingungen:	27
2.2.2	Blocklehrveranstaltungen	30
2.3	Ziel	31
2.4	Ein erläuterndes Beispiel	33
3	**Umsetzung mit ILOG/OPL**	**35**
3.1	Struktur	35
3.2	OPL	36
3.3	Beschreibung und Funktionsweise des Modells in OPL	36
3.3.1	Beschreibung der Daten	37
3.3.2	Informationsfluss	38
3.3.3	Beschreibung des Programmcodes für das Modell	39

3.3.3.1 Umformen der Daten in Vektoren . . . 39
3.3.3.2 Konsistenzprüfung der Daten . . . 39
3.3.3.3 Deklaration der Variablen . . . 40
3.3.3.4 Implementation der Randbedingungen . . . 40
3.3.3.4.1 Gleichzeitigkeit der Vorlesungen . . . 41
3.3.3.4.2 Mehrteilige Vorlesungen . . . 42
3.3.3.4.3 Wahl des passenden Hörsaals . . . 43
3.3.3.4.4 Thematische Überschneidungen verhindern . . . 47
3.3.3.4.5 Zeitlichen Bereich der Vorlesungen einschränken . . . 49
3.3.3.4.6 Ein zusätzlicher Raum . . . 49
3.3.3.4.7 Wünsche . . . 50
3.3.3.4.8 Erzwungene Zeiten und Orte – Pflichten/Verbote . . . 53
3.3.3.4.9 Voneinander unabhängige Vorlesungen koppeln . . . 54
3.3.3.4.10 Blocklehrveranstaltungen . . . 59
3.3.3.4.11 Gemeinschaftlich gehaltene Vorlesungen . . . 61
3.3.3.4.12 Verfügbarkeit der Vortragenden . . . 64
3.3.4 Strategie . . . 66
3.3.5 Behandlung von Widersprüchen - „Soft-Constraints“ . . . 68
3.4 Lösung . . . 72
3.4.1 Tests . . . 74

4 Handbuch zum Vorlesungsplaner 77
4.1 Installation . . . 77
4.2 Arbeits- und Konfigurationsdateien . . . 79
4.3 Bedienung . . . 79
4.3.1 Schritt 1 - Basiskonfiguration . . . 79
4.3.2 Schritt 2 - Dateneingabe . . . 80
4.3.2.1 Grundlegendes . . . 80
4.3.2.2 Hörsäle . . . 81
4.3.2.3 Vortragende . . . 81
4.3.2.4 Vorlesungen . . . 81
4.3.2.5 Restliche Daten . . . 82
4.3.3 Schritt 3 - Bedingungen an den Vorlesungsplan . . . 82
4.3.3.1 Fixe Vorgaben . . . 83
4.3.3.2 Notwendige Randbedingungen . . . 83
4.3.4 Schritt 4 - Wünsche . . . 84
4.3.5 Schritt 5 - Lösung und Ausgabe . . . 84
4.3.5.1 Auffinden von Widersprüchen . . . 84
4.3.6 Schritt 6 - Backup . . . 85
4.3.7 F. A. Q. . . . 85

5 Dokumentation des Programmes 95
5.1 Aufbau . . . 95
5.1.1 Webseiten . . . 95
5.1.2 Struktur . . . 96
5.1.3 Sicherheit . . . 96
5.1.4 Datenverwaltung . . . 96
5.2 Entwicklungsablauf . . . 97
5.2.1 Planung und Organisation . . . 97
5.2.2 Versionsmanagement . . . 98
5.2.3 Backupmanagement . . . 99

6 Zusammenfassung und Ausblick 101

7 Source-Code des Modells 103
7.1 `zeitplanung.mod` 103
7.2 `zeitplanung.dat` 121
7.3 `zeitplanung.xml` 124
7.4 Daten von WS2005/06 128
7.5 Lösung zu WS2005/06 140

Literaturverzeichnis 143

Lebenslauf 147

0 Einleitung

Der mathematische Zweig „Angewandte Mathematik“ versucht praktisch auftretende Probleme mit mathematischen Methoden zu lösen. Alle Teilgebiete der Mathematik werden herangezogen, um ein Problem formal zu übersetzen und eventuell mit einem Computer zu lösen. Diese hier vorgestellte Anwendung in Form eines Vorlesungsplaners kann an einer Universität verwendet werden, um Vorlesungen, Hörsäle und Vortragende zu managen.

Das Programm verwendet diskrete Objekte und Zustände, welche ihr einen kombinatorischen Charakter verleihen. Es ist so aufgebaut, dass es über eine durch einen Webbrowser erreichbare Schnittstelle alle Daten sammelt und anschließend mittels eines externen Solvers versucht, eine Lösung zu berechnen. Ziel ist es, den Stundenplan für die Vorlesungen eines Semesters automatisch zu generieren. Dafür wird ein Modell herangezogen, welches alle Bedingungen für einen funktionierenden und zufriedenstellenden Stundenplan kennt und in einem System von logischen Bedingungen ausdrückt. Außerdem müssen Daten zu den Vorlesungen, Vortragenden und Hörsälen erfasst werden, um damit Berechnungen anstellen zu können. Hinzu kommen Bedingungen, die viele – wenn nicht alle – in der Praxis auftretenden Einschränkungen erfassen können:

- Blocklehrveranstaltungen,
- von mehreren Vortragenden gehaltene Vorlesungen,
- näher und weiter entfernte Räumlichkeiten einberechnen,
- thematische Vorlesungsgruppen (Anfänger, Seminare, spezialisierte Vorlesungen...),
- zeitliche Einschränkungen (an Hörsäle und Vortragende),
- paarweise Verknüpfungen zwischen Vorlesungen (ausschließend und verschränkend),
- Nutzung des PC-Labors,
- zeitliche Reihungen und Nachfolger,
- Hörsaalgrößen mit dazu passender Wahl der Vorlesungen, (d.h. immer genügend Sitzplätze bei großen und keine kleinen Vorlesungen in großen Hörsälen).

Darüber hinaus können von Vortragenden und Studenten Wünsche an die Einteilung geäußert werden.

Die Ausgabe der Daten erfolgt wahlweise als HTML-Webseite oder PDF-Dokument. Um die Daten nicht für jedes Semester erneut eingeben zu müssen, kann der Systemadministrator des Servers die in einer XML-Datei hinterlegten Daten archivieren und nach Bedarf die momentan verwendete auswechseln.

Die hier vorliegende Diplomarbeit beschreibt zuerst die theoretischen Hintergründe, welche Ansätze verwendet werden um Probleme dieser Art behandeln zu können. Dann wird ein rein mathematisches Modell erarbeitet, dessen Umsetzung anschließend detailliert erläutert wird. Auch werden Alternativen und weiterführende Ideen aufgezeigt. Ein Handbuch für Anwender, welches die Bedienung und Konzepte aus deren Sicht beschreibt, ist ebenso wie eine Installationsanleitung für den Systemadministrator vorhanden. Das Programm ist auch beschrieben und dokumentiert, um daran Verbesserungen vornehmen zu können.

Im Herbst 2005 habe ich angefangen, an dieser Diplomarbeit zu arbeiten. Damals wurde im Groben festgelegt, ein funktionierendes Programm als Hilfe zur Erstellung des Vorlesungsplanes zu entwickeln. Begonnen habe ich, nach einigen grundlegenden Ideen, mich in das Handbuch des ILOG Solvers einzuarbeiten. Danach entwickelte ich in mehreren Schritten Entwürfe für Modelle, die Teile des Gesamtproblems abdecken. Auch lernte ich so mit dem Solver umzugehen und sammelte wichtige Erfahrungen. Bis zum Ende des Jahres 2005 hatte ich ein weitgehend fertiges Modell und begann an der Arbeit für das Interface. Da ich noch nie etwas derartiges programmiert hatte, eignete ich mir in zwei Wochen das nötige Basiswissen für den Umgang mit dem Tomcat Server und Java an. Im ersten Drittel des Jahres 2006 hatte ich zwei rudimentäre Module programmiert, eines für das Interface und eines für die Anbindung des Solvers, welche nun verbunden werden konnten. Nach einigen Tests wurden mir schnell einige grundlegende Probleme bewusst, die es notwendig machten, das Programm teilweise neu zu schreiben. Die Anbindung des Solvers machte ebenfalls Schwierigkeiten, welche ich erst im Herbst 2006 in den Griff bekam. Parallel zur Entwicklung des Interfaces Anfang 2006, gab es noch einige Ergänzungen zum im Herbst davor entwickelten Modell. Beispielsweise wurden die Wünsche der Vortragenden und die Blocklehrveranstaltungen implementiert. Im Sommer 2006 war ich mit Fehlersuche und Tests beschäftigt. Da das Programm bereits zu dieser Zeit auf über 10000 Zeilen angewachsen war, war dies eine recht umfangreiche Aufgabe. Im darauf folgenden Wintersemester 2006/07 begann ich die parallel geführten Aufzeichnungen für die Diplomarbeit zu einem Ganzen zusammenzuschreiben. Im Winter 2006 verfasste ich dann den Abschnitt über den theoretischen Hintergrund und konnte die Diplomarbeit im Frühjahr 2007 in eine fertige Fassung bringen.

Für Unterstützung bedanken möchte ich mich bei meinem Betreuer Hermann Schichl und meinen Eltern, die mir mein Studium ermöglicht haben.

1 Hintergrund

Diese Diplomarbeit beschreibt die Berechnung eines Zeitplanes für Vorlesungen an einer Universität. Das ist ein spezielles Problem aus dem Gebiet der kombinatorischen Optimierung. Hierbei werden Probleme behandelt, deren Beschreibung diskrete Variablen verwenden. Konkret heißt das, dass als Wertebereich kein kontinuierlicher Bereich der reellen Zahlen verwendet wird, sondern mit klar getrennten Zuständen gearbeitet wird.

Die kombinatorischen Optimierung ist eine relativ junge Disziplin. Es gibt vergleichsweise einfach zu formulierende Problemstellungen, die schon seit Jahrhunderten bekannt sind. Befassen sie sich nur mit den Zahlen an sich, so führt dies zu den Gebieten der Zahlentheorie und Kombinatorik. Hier werden Probleme, die sich als kombinatorisches oder zahlentheoretisches Problem formulieren lassen behandelt, indem eine Theorie um sie entwickelt wird. Das gelingt dadurch, indem – wie in der Geschichte der Mathematik schon oft so geschehen – Zusammenhänge und Strukturen erkannt und auf einer allgemeineren Ebene behandelt werden.

Das Gebiet der Optimierung ist viel jünger und behandelt vornehmlich Probleme, welche im allgemeinen versuchen, durch Variieren der Werte eines Vektors von Variablen den Wert einer Funktion – der Zielfunktion – zu maxi- oder minimieren. Kombinatorische Optimierung steht im Spannungsfeld zwischen einerseits Kombinatorik und Zahlentheorie und andererseits Optimierung. Es ist häufig so, dass es zusätzliche Bedingungen an die Variablen gibt. Diese dürfen nicht vollkommen frei gewählt werden, sondern müssen zusätzlich eine Liste von logischen Aussagen erfüllen.

Das hier verwendete kombinatorische Optimierungsproblem hat folgende Struktur:

$$\begin{aligned} \text{maximiere} \quad & c(x) \\ \text{s.t.} \quad & x \in \mathbf{X}. \end{aligned} \tag{1.1}$$

Gleichung 1.1 ist so zu verstehen, dass $c(x)$ eine von den Variablen abhängige Funktion ist, deren Wert maximiert werden soll. Dies lässt sich so verstehen, dass die einzelnen für x zulässigen Werte eine Bewertung bekommen. Die Bedingungen an den Vektor der Werte ist hier ganz allgemein so gegeben, dass sie nur aus einer diskreten Menge $\mathbf{X}$ gewählt werden dürfen.

In der Praxis zeigt sich, dass es hier zwei grundlegende Probleme gibt. Erstens ist es kompliziert, den Wertebereich $\mathbf{X}$ anzugeben. Dieser beschreibt den abzusuchenden Lösungsraum und beinhaltet die gesamte Information über die Art des Problems. Zweitens sind hier für x nur zulässige Werte erwünscht. Dies führt zu der Frage, ob es bei einer großen Anzahl von Bedingungen an x überhaupt zulässige Werte gibt und wenn ja, welche

es sind. Das sind die beiden für diese hier behandelte Art von Fragestellung maßgeblichen Problemstellungen.

Wie schon erwähnt, ist die kombinatorische Optimierung relativ jung. Die Entwicklung der Mathematik hat gezeigt, dass am Anfang der Entwicklung vieler Teildisziplinen zuerst hauptsächlich mit heuristischen Methoden, also geschicktem Probieren und Austesten von Ideen, gearbeitet wurde. Es gibt daher keine alles umfassende einheitliche Theorie, sondern einige Ansätze und Ideen, welche sich in der Praxis hilfreich erweisen. Im folgenden werden Methoden erörtert, welche auf diese Probleme angesetzt werden können.

1.1 Problembehandlung

Um so ein Problem wie Gleichung 1.1 behandeln zu können, ist es notwendig es umzuformulieren. Das bedeutet, es wird eine alternative Beschreibung desselben Problems gesucht, welche auf dieselbe – oder beinahe dieselbe – Lösung führt. Das ist notwendig, damit dieses Problem mit einer der zur Verfügung stehenden Methoden behandelt werden kann.

Um besser damit arbeiten zu können, führe ich für dieses Kapitel ein eigenes kleines Zeitplanungsbeispiel ein. Daran lassen sich die behandelten Methoden vorzeigen und gut erklären. Das dann tatsächlich verwendete Modell ist um ein vielfaches komplexer.

Das Beispiel besteht aus drei Zeitspannen, welche zur Vereinfachung gleich lang dauern und jeweils nur an vier zeitlich aufeinanderfolgenden Positionen eingeteilt werden können. Diese einzuteilenden Zeitspannen stehen für die Vorlesungen, welche ebenfalls eine fixe Zeitspanne dauern (wenn auch nicht immer dieselbe, jedoch von diskreter Länge sind) und deren Startzeitpunkte im Zuge der Optimierung gewählt werden müssen. Die vier möglichen Positionen stehen genauso wie im Fall der Stundenplanung für die begrenzte Dauer eines Tages, in dem die Vorlesungen eingeteilt werden. Darüber hinaus werden an diese Zeitspannen Bedingungen gestellt. Eine von ihnen gibt an, dass nur Positionen ab einem bestimmten Wert gewählt werden können – dies entspricht in dieser Vereinfachung der Bedingung, dass eine Vorlesung erst ab einem fixen Zeitpunkt gehalten wird. Eine weitere zwingt zwei Zeitspannen, zeitlich aufeinander zu folgen. Die dritte Bedingung gibt schließlich an, dass keine der drei Zeitspannen zur selben Zeit stattfinden darf – dies entspricht der Tatsache, dass in einem Hörsaal zur selben Zeit nur eine Vorlesung stattfinden kann.

Hier die Formulierung:

$$
\begin{aligned}
&(A, B, C) \in \{1, 2, 3, 4\}^{(3)} \\
&\quad \text{(C1)} \quad B \geq 3 \\
&\quad \text{(C2)} \quad A > B \\
&\quad \text{(C3)} \quad A \neq B \neq C, A \neq C.
\end{aligned}
\tag{1.2}
$$

A, B und C stehen hierbei für die drei Zeitspannen, wobei ihre Werte die Zeitpunkte 1 bis 4 sein können. C1 bis C3 sind die drei oben erwähnten Randbedingungen. Zielfunktion

gibt es vorläufig keine, da es in erster Linie wichtig ist, überhaupt eine gültige Belegung zu finden.

Um dies bewerkstelligen zu können, ist es sinnvoll, eine Funktion einzuführen, die den Grad der Verletzung der drei Randbedingungen angibt. Da jede der Bedingungen wahr oder falsch sein kann, bietet sich an, ihnen die Werte 0 oder 1 zuzuweisen. Da Bedingung C3 in Wahrheit aus mehreren Bedingungen besteht, die teilweise richtig und falsch sein können, sollte sie in drei Unterbedingungen aufgespalten werden. Diese bekommen dann zu gleichen Teilen jeweils den Wert $1/3$ wenn sie wahr sind.

Hier diese Erweiterung:

$$\begin{aligned} &(A, B, C) \in \{1, 2, 3, 4\}^{(3)} \\ &\textbf{(C1)} \quad B \geq 3 \\ &\textbf{(C2)} \quad A > B \\ &\textbf{(C3.1)} \quad A \neq B \\ &\textbf{(C3.2)} \quad A \neq C \\ &\textbf{(C3.3)} \quad B \neq C. \end{aligned} \tag{1.3}$$

Eine dazugehörige Funktion c für den Grad der Verletzung dieser Nebenbedingungen sieht so aus:

$$c((A, B, C)) = [B \geq 3] + [A > B] + \tfrac{1}{3}([A \neq B] + [A \neq C] + [B \neq C]). \tag{1.4}$$

wobei die Ausdrücke in eckigen Klammern je nach Wahrheitswert entweder zu 0 oder 1 ausgewertet werden.

Eine Belegung $(A, B, C) = (2, 1, 2)$ hat den Wert $c((2, 1, 2)) = \frac{5}{3}$, da B nicht größer gleich 3, A größer als B und nur zwei der drei Ungleichheitsbedingungen erfüllt werden.

Ausgerüstet mit dieser Bewertungsfunktion 1.4 lassen sich Suchalgorithmen beschreiben. Sie alle behandeln diese Art von kombinatorischen Problemen, indem sie versuchen alle Randbedingungen zu erfüllen. Im Normalfall wird eine große Menge von Werten gefunden, die sich wiederum durch eine Zielfunktion bewerten lassen.

Handelt es sich zum Beispiel um eine Zeitplanung mehrerer Aufgaben, so ist es häufig von Interesse, die Gesamtzeit zum Erledigen der Arbeit zu minimieren. Klassischer Anwendungsfall ist das Erbauen eines Hauses, welches aus vielen einzelnen Tätigkeiten besteht, die untereinander Abhängigkeiten besitzen (Fenster können erst eingesetzt werden, wenn die Wände vorhanden sind, etc.). Alternativ könnten auch die Kosten wesentlich sein, oder eine Kombination mehrerer Kriterien.

1.2 Suchalgorithmen

Jeder Suchalgorithmus versucht Variablenbelegungen zu finden, die die Randbedingungen (möglichst gut) erfüllen.

Definition 1.1. *Ein Suchalgorithmus ist ein endliches, algorithmisch beschreibbares Verfahren, welches für den in Problem 1.1 gesuchten Vektor x eine im Sinne der Zielfunktion möglichst gute Lösung zurückgibt.*

1.2.1 Komplexität

Theoretisch lässt sich leicht die beste Lösung finden, indem alle möglichen Kombinationen durchprobiert werden. Bei dem hier verwendeten stark vereinfachten Beispiel 1.2 ist dies noch machbar – in der Praxis jedoch unmöglich. Das liegt daran, dass der Suchraum i. a. exponentiell mit der Anzahl der Variablen wächst aber der zur Verfügung stehende Zeitrahmen klein bleibt. Das Ziel ist es also, ein Minimum oder Maximum für eine in polynomialer Zeit berechenbare Kostenfunktion c zu finden. Das ist die Komplexitätsklasse NPO nach [ACG+99].

Bei dem später behandelten Modell zur Einteilung der Vorlesungen werden Variablen in der Größenordnung von etwa Tausend mit jeweils zwischen 5 und 96 möglichen Zuständen vorkommen. Das sind klarerweise viel zu viele Möglichkeiten um sie jemals durchprobieren zu können. Daher muss man mit möglichst effizienten Methoden arbeiten, um schon nach einem sehr kleinen Bruchteil aller Kombinationsmöglichkeiten ein verwertbares Ergebnis zu erhalten.

Das führt zu den sogenannten *Heuristiken*, die sich in der Praxis als nützlich erwiesen – jedoch theoretische Aufarbeitung benötigen. Deshalb gilt es in der Zukunft daran zu arbeiten, diese Methoden besser zu untersuchen, aus ihnen allgemeine Schlussfolgerungen zu ziehen und sie auf eine solide theoretische Basis zu stellen.

1.2.2 Simulated Annealing

Eine einfache Idee ist, zwei zufällige Belegungen für die Variablen zu finden. Dann wird diejenige Belegung genommen, welche die bessere Bewertung durch die Funktion c bekommt. Das führt zur Methode des „Simulated Annealing“ (siehe [KGV83]).

Sie funktioniert so, dass es einen zufälligen Startwert gibt. Dieser wird laufend um einen kleinen, zufälligen Wert verändert. Es wird ein Wert x_{neu} durch $s(x_{alt})$ berechnet, wobei s eine Störfunktion ist.

Jedes Mal wird die Bewertungsfunktion ausgerechnet und entweder ein neuer bester Wert für x akzeptiert, oder falls es keine Verbesserung gab, dieser schlechtere Wert mit einer im Lauf der Optimierung sinkenden Wahrscheinlichkeit trotzdem als neuer Wert für x angenommen.

Listing 1.1: Simulated Annealing

```
1 initialisiere x
```

```
2  w <- c(x)
3  t <- 0
4  iteriere
5          x_neu <- s(x)
6          wenn(c(x_neu) > c(x))
7                  dann x <- x_neu
8          sonst   wenn(a(x,x_neu,t)) dann x <- x_neu
9          t <- t + 1
10 solange bis abbruch(t)
```

Hierbei ist `s` in Listing 1.1 die Störfunktion, die zum Beispiel einige wenige der Variablen um einen kleinen Wert zufällig ändert. Die Funktion `a` berechnet aus der Anzahl an Iterationen und den Werten der neuen und alten Variablenbelegung einen Wahrheitswert. Ist dieser wahr, so wird auch ein schlechterer Wert von $c(x)$ akzeptiert. Sie besteht häufig, in Anlehnung an den physikalischen Prozess einer Abkühlung – der ihr Namenspatron ist – aus einer streng monoton fallenden Exponentialfunktion und eventuell zusätzlich aus der Differenz von $c(x_{neu}) - c(x)$.

Letzteres hat den Sinn, um mit einer gewissen geringen Wahrscheinlichkeit aus einem lokalen Minimum ausbrechen zu können. Ist die Differenz $c(x_{neu}) - c(x)$ oder auch relative Differenz $\frac{c(x_{neu})-c(x)}{c(x)}$ klein, so ist der dadurch begangene Verlust an Optimalität gering. Umgekehrt sollte die Wahrscheinlichkeit des Akzeptierens schlechterer Belegungen mit Fortschreiten der Optimierung sinken, um ständiges Ausbrechen aus einem lokalen Minimum zu vermeiden.

`abbruch` ist eine Abbruchbedingung. Hier ist sie nur von der Anzahl der Iterationen `t` abhängig. Sie kann jedoch auch von den zuletzt erzielten Verbesserungen der Bewertung $c(x)$ abhängig sein.

Nicht klar definiert ist die Störfunktion `s`. Ihr Design kennzeichnet die genaue Qualität der Implementation und muss für jeden Anwendungsfall im Zusammenspiel mit den anderen Funktionen getestet werden.

Durch Parallelisierung lässt sich die Effizienz steigern. An den Stellen, wo die Funktion `a` erreicht wird, ließe sich eine Verzweigung in zwei unabhängige Verarbeitungsprozesse einleiten. Ein Prozess arbeitet weiterhin mit der bisher besten Lösung und ein anderer mit einer neuen etwas schlechteren Lösung. Eine mögliche Strategie wäre, abzuwarten ob sich im neuen Prozess bald ein besserer Wert für c finden lässt. Ist dem so, so kann der alte beste Wert verworfen werden und der dazugehörige Subprozess wird beendet. Anderenfalls wird mit dem bisher besten Wert weitergearbeitet und eventuell erneut verzweigt. Mit dieser Maßnahme sinkt die Gefahr, gute Lösungen zu verlieren ohne auf die Chance auf bessere Lösungen verzichten zu müssen. Freilich gibt es mehrere Möglichkeiten der Parallelisierung, drei werden bei [AK89] beschrieben. Ein Modell zur Berechnung eines Stundenplanes einer Schule mit Parallelisierung wird in [Abr91] erklärt.

Eine weitere Verbesserung besteht darin, den Einfluss auf Änderungen der Variablen zu untersuchen. Reagiert die Funktion c auf bestimmte Variablenänderungen sehr empfindlich, so sollten diese nicht so häufig verändert werden um starke Fluktuationen zu vermeiden. Auch bietet sich hier an, den Variablenvektor in Teilmengen aufzutrennen. Es

wird dann nur mit einem Teil der Variablen gearbeitet, während die anderen fix gehalten werden. Ist ein lokales Extremum erreicht, wird auf einer anderen Teilmenge weitergearbeitet. Es ist klarerweise vom Geschick des Programmierers abhängig, Algorithmen zu entwickeln, die diese Teilmengen erkennen. Die Struktur der Randbedingungen könnte dabei einen Hinweis geben.

Ein anderer Ansatz besteht darin, direkt den zufälligen Startvektor passend zu wählen. Diverse Probleme kombinatorischer Natur lassen es zu, dass eine ihrer Bedingungen unter Missachtung aller anderen explizit zur Berechnung einer möglichen Konstellation verwendet werden kann. In unserem Fall ist das zum Beispiel Bedingung `(C3)` aus den Randbedingungen 1.2. Es ist erstens klar, dass es mehr Zeitfenster als zuzuweisende Blöcken geben muss. Zweitens kann der Startvektor so generiert werden, dass eine zufällige Belegung nach der anderen erzeugt wird, wobei darauf geachtet wird, dass kein schon verwendetes Zeitfenster erneut benützt wird. Zwar könnte dies im Zuge der Optimierung wieder verletzt werden, für die Praxis zeigten sich solche Ansätze in verwandten Problemen als nützlich (*Traveling-Salesperson-Problem*, [BL84]). Das liegt daran, dass wegen der geringen Änderungen durch die Störfunktion hauptsächlich lokal gesucht wird und daher der Sucherfolg stark vom Startwert abhängt.

Eine andere Verbesserung ist die bei [DS90] beschriebene, eine schlechtere Lösung nur dann zu akzeptieren, wenn sie wie oben von einer stetig fallenden Wahrscheinlichkeit abhängt und die Zielfunktion höchstens um einen ebenfalls fallenden, vorher festgelegten Wert verletzt. Dies verhindert zufällige sehr große Störungen und ist für die Praxis sehr nützlich.

1.2.3 Tabu Search

Bei Simulated Annealing, beschrieben in [Glo86], wird von einem momentanen Wert aus mittels einer Störfunktion s ein diesem benachbarter Wert untersucht. Dieses Nachbarschaftsverhältnis zeichnet sich dadurch aus, dass es nur eine geringe Änderung zwischen diesen beiden Werten gibt. Die Methode der „Tabu Suche“ ist ein Versuch, dieses Absuchen der Umgebung zu systematisieren. Ihre Grundidee besteht darin, auf eine etwas aggressivere Art als die des „Simulated Annealing“ einen neuen besseren Wert zu finden. Dabei wird jedoch nicht das gesamte benachbarte Gebiet untersucht, sondern es gibt Zonen, die nicht gewählt werden dürfen – diese Bereiche sind „tabu“. Diese Bereiche werden dynamisch gewählt und passen sich dem laufenden Suchprozess an. Die dahintersteckende Motivation besteht darin, exzessives Absuchen kleiner lokaler Stellen zu vermeiden, um eine größere Zerstreuung des Suchpfades zu erreichen.

Es gibt somit zwei parallel mitlaufende Datentypen, die den Suchverlauf protokollieren. Erstens werden die zuletzt gewählten Werte für Variablen für eine kurze Zeit gespeichert – zweitens die Häufigkeit einer bestimmten Wertänderung. Aus diesen Informationen wird entschieden, welche Werte als nächstes untersucht bzw. zugewiesen werden dürfen.

Es ist auch diesmal nicht genau festgelegt, was ein Nachbarschaftsverhältnis ist. Dieses muss jedoch so beschaffen sein (siehe [Ree95]), dass

1. von jedem Wert einer Variablenzuweisung aus nach endlich vielen Schritten jeder

andere gewählt werden kann, damit das Gebiet zusammenhängend ist,

2. und wenn $x' \in N(x)$ Nachbar von x ist, so ist auch $x \in N(x')$ Nachbar von x' (Symmetrie).

Nach [Ree95] ist die Tabusuche aber durchaus praktikabel, wenn der zweite Punkt nicht immer erfüllt ist.

Der Algorithmus läuft iterativ, und es gibt zusätzliche Kriterien, die auf Basis heuristischer Annahmen die Dynamik des Suchvorganges beeinflussen.

Neu ist die Matrix H. Sie speichert den Verlauf der bisherigen Suche und ist Basis für die Berechnung der Nachbarschaftsumgebung und die Bewertung der neuen besten Lösung. Zwei Variablenbelegungen sind dann Nachbarn, wenn nur eine kleine Änderung der Werte zwischen ihnen steht. Das kann die Änderung eines Wertes in einem Vektor sein, das Vertauschen von zwei Elementen in einer geordneten Liste oder ein anderer definierter Übergang. Diese Matrix ist über die einzelnen Zustände indiziert und teilt sich in zwei Dreiecksmatrizen. Die obere Dreiecksmatrix merkt sich die letzten Änderungen für eine festgelegte Anzahl von Iterationen. Das heißt, sie vergisst nach einigen Durchgängen die Einträge. Das lässt sich so programmieren, dass eine Zahl an die Stelle (a, b) eingetragen wird, welcher für den Übergang der Variablen von a nach b oder b nach a steht. Sollen die Änderungen für drei Iterationen gespeichert bleiben, so wird eine 3 eingetragen. Nach jeder Iteration wird die Matrix H aktualisiert. Dabei wird jeder dieser Einträge um 1 reduziert und Einträge die 0 ergeben gelöscht. Die untere Dreiecksmatrix zählt die Häufigkeiten der einzelnen Übergänge. Es wird für einen Übergang zwischen a und b an der Stelle (b, a) eine 1 zu dem Wert an dieser Stelle addiert – ist noch nichts gespeichert, so wird eine 1 eingetragen. Hat eine Variable vier Zustände, so kann nach einigen Iterationen die Matrix H so aussehen:

	1	2	3	4
1	■	1	2	
2	2	■		
3	1	1	■	3
4		2		■

Tabelle 1.1: Matrix H

Hier ein Algorithmus mit verschiedenen Ausprägungen des zweiten Schrittes, um die Tabusuche in einem größeren Kontext zu sehen. `H` bezeichnet dabei die oben eingeführte Matrix über den bisherigen Suchverlauf. Durch diese Matrix wird die Nachbarschaft (`N`) von einem aktuellen Wert `x` laufend verändert. Das charakterisiert die Tabusuche.

Listing 1.2: Nachbarschaftssuchmethoden

```
1 initialisieren von x \in X
2
3 Schritt 1:
4         x_best <- x
5         c_best <- c(x_best)
```

```
6
7  Schritt 2:
8          [Descent Method]
9                  waehle x_neu aus N(x), wobei c(x_neu) < c(x) gelten muss
10                 sonst abbruch
11         [Monte Carlo Method]
12                 (a) waehle x_neu zufaellig aus N(x)
13                 (b) ist c(x_neu) <= c(x), gehe zu Schritt 3
14                 (c) ist c(x_neu) > c(x) so akzeptiere x_neu mit einer
15                     Wahrscheinlichkeit, die mit steigender Differenz
16                     c(x_neu) - c(x) faellt.
17                     Wird x_neu nicht akzeptiert, gehe zu (a)
18                 (d) teste Abbruchkriterium
19         [Tabu Search]
20                 (a) x_Kandidaten Teilmenge von N(H,x)
21                 (b) x_neu <- waehle x aus x_Kandidaten, welches c(H,x) minimiert
22                 (c) aktualisiere H
23                 (d) teste Abbruchbedingung
24 Schritt 3:
25         x <- x_neu
26         wenn c(x) < c_best gehe zu Schritt 1, sonst Schritt 2
```

Dieser Algorithmus 1.2 ist vom zufälligen Startwert abhängig. Daher lohnt es sich, verschiedene Startwerte zu verwenden. Es kann probiert werden, so lange zufällige Startwerte zu nehmen, bis ein relativ zu den anderen guter Wert gefunden wird und erst dann mit dem Algorithmus starten. Weiters lässt sich der Algorithmus mehrmals aufrufen und die unterschiedlichen Ergebnisse miteinander vergleichen. Dabei kann das Abbruchkriterium nicht nur von der aktuellen Zahl der Iterationen, sondern auch zusätzlich vom bisher besten Wert der Zielfunktion abhängen. Dadurch wird vermieden, unnötig lange in einem „schlechten“ lokalen Minimum zu verweilen. Dies lässt sich auch parallelisiert implementieren, wobei jeder Subprozess in dessen Abbruchbedingungen Zugriff auf die momentan besten Werte hat und so die Qualität seines eigenen momentanen Suchvorganges bewerten kann.

1.2.3.1 Tabukriterien

Ein Zug kann aus mehreren Gründen „tabu“ sein. Die Idee ist, dass exzessives Absuchen eines kleinen Bereichs des Suchraumes verhindert werden soll. Daher dürfen sich aufeinanderfolgende Änderungen nicht gegenseitig aufheben. Um das zu erkennen, gibt es mehrere Überlegungen. Eine ist, dass auf eine Änderung einer Variable nicht gleich wieder das Zurücksetzen auf ihren vorhergehenden Wert folgen darf. Dies kann verfeinert werden, indem die Züge als Änderung von einem Wert auf einen anderen definiert werden. Dann kann ein Tabukriterium darin bestehen, dass das Gegenteil zu einem Zug in der Liste der letzten Züge nicht aufscheinen darf. Ein anderes ist etwas strenger und besagt, dass die Kombination zweier Werte nicht wieder vorkommen darf – das bedeutet, nicht nur der eine Zug der dies ergibt, sondern auch eine Kette von Zügen die wieder den alten

Zustand erzeugen könnte ist tabu. Es können Reduktionsfunktionen verwendet werden, die aus der Variablenbelegung nur Teile herausnehmen oder Belegungen zusammenfassen. Diese können dann dazu verwendet werden, Züge auszuschließen bei denen der Wert dieser Funktion wieder gleich dem in der Vergangenheit war.

Zusammenfassend gesehen, werden ständige Wiederholungen von Wertzuweisungen und Rücksetzen auf schon kürzlich dagewesene Zustände verhindert.

1.2.3.2 Aspirationskriterien

Die *Aspirationskriterien* sind eine Ergänzung der Tabusuche. Sie erlauben, einen Wert für die Variablen anzunehmen, auch wenn er „tabu“ ist. Dies kann auf Basis mehrerer Entscheidungskriterien passieren.

Eines wäre die Tatsache, dass eine Belegung genommen wird, wenn der Wert der Zielfunktion höher als der beste bisherigen Wert ist. Das setzt natürlich voraus, dass anstatt nur der erlaubten immer alle Werte getestet werden müssen. Das ist ein gewisser Mehraufwand.

Ein triviales Kriterium besteht darin, dass wenn alle möglichen Züge verboten sind, ein „am wenigsten verbotener“ genommen wird.

Da es mehrere Möglichkeiten gibt zu definieren, welche Züge tabu sind, kann diese Liste noch um andere Kriterien erweitert werden.

1.2.3.3 Beispiel

Zur Demonstration behandle ich Beispiel 1.2 von Seite 4 mit der Tabusuche. Es wird weiters die schon bekannte Bewertungsfunktion 1.4 c verwendet. Die Nachbarschaft zwischen zwei Variablenbelegungen definiere ich so, dass sich einer der drei Werte für A, B und C ändert. Zusätzlich gelten einige Regeln, auf die oben eingegangen wurde.

Schritt 0 Die Variablen (A, B, C) werden mit $(1, 1, 1)$ initialisiert. Der Wert von c ist 0, da alle Bedingungen verletzt werden.

Schritt 1 Nun Startet der Tabu Search Algorithmus. Es werden alle neuen Belegungen für die Variablen A, B und C bewertet. Da es noch keine ausgeschlossenen Belegungen gibt, kann gleich die beste genommen werden. Diese wird mit einem „*“ gekennzeichnet. Auch mache ich es nicht parallel, sondern zuerst für Variable A, dann für B und dann für C.

Abbildung 1.1 beinhaltet die Tabellen nach dem ersten Schritt.

A	1	2	3	4	Wert
1	■	3			-
2	1	■			5/3 *
3			■		5/3
4				■	5/3

(a) Variable A

B	1	2	3	4	Wert
1	■		3		-
2		■			1/3
3	1		■		2 *
4				■	2

(b) Variable B

C	1	2	3	4	Wert
1	■			3	-
2		■			5/3
3			■		5/3
4	1			■	2 *

(c) Variable C

Abbildung 1.1: Tabu Search – Schritt 1

Die beste und erste Wahl für A ist der Wechsel von 1 auf 2. Anschließend ergeben die Bewertungen für B, dass ein Wechsel von 1 auf 3 c verbessert. Für C ergibt sich, ausgehend von $(2, 3, 1)$, für die Variablenbelegung von C, dass 1 auf 4 die beste Wahl ist – hier verbessert sich die Zielfunktion nicht, wird jedoch im Vergleich zu den anderen Möglichkeiten nicht schlechter.

Zusammengefasst sind wir nun bei der Belegung $(A, B, C) = (2, 3, 4)$.

Schritt 2 Der zweite Schritt beginnt wieder bei A. Der Wechsel von 2 auf 1 würde Schritt 1 zurücknehmen. Daher ist er jetzt ausgeschlossen und trägt in der Tabelle das **T**.

B ist interessanter. Alle Änderungen verschlechtern die Situation, außer es wird nichts verändert. Da es sich außerdem um den besten bislang erreichten Wert für c handelt, wird 3 beibehalten (Aspirationskriterium). Das maximale Ergebnis für c wird dann mit $C = 2$ erreicht.

Abbildung 1.2 beinhaltet die Tabellen nach dem zweiten Schritt.

A	1	2	3	4	Wert
1	■	2			2 **T**
2	1	■		3	-
3			■		5/3
4		1		■	8/3 *

(a) Variable A

B	1	2	3	4	Wert
1	■		2		5/3 **T**
2		■			5/3
3	1		■		8/3 *
4				■	1

(b) Variable B

C	1	2	3	4	Wert
1	■			2	- **T**
2		■		3	3 *
3			■		8/3
4	1	1		■	8/3

(c) Variable C

Abbildung 1.2: Tabu Search – Schritt 2

Damit sind alle drei Randbedingungen 1.2 erfüllt und $(A, B, C) = (4, 3, 2)$ die erste gefundene Lösung.

Damit hat dieser Algorithmus nach 6 Schritten, eine Lösung gefunden, wobei es im gesamten Suchraum $4^3 = 64$ Möglichkeiten gibt.

1.2.3.4 Verbesserungen

Es ist schnell einsichtig, was die Schwächen von Tabu Search sind. Randbedingungen werden genauso wie die Zielfunktion behandelt. Das heißt, diese Methode ist gut darin, schnell einen guten Wert für die Zielfunktion zu erhalten. Jedoch versagt sie, wenn alle Randbedingungen erfüllt werden müssen. Deshalb eignet es sich nicht so gut, um eine vollständig richtige Lösung zu finden.

Natürlich kommt es darauf an, wie das Problem behandelt wird. Bei größeren Problemstellungen ist es durchaus erlaubt, einige weniger wichtige Randbedingungen zu verletzen, wenn es im Ausgleich dazu überhaupt eine Lösung gibt!

Besser wäre es, wenn das Verfahren genauer auf die individuellen Eigenschaften der Randbedingungen eingehen würde. Das führt zur Methode der „Constraint Propagation“.

1.2.4 Constraint Propagation

Constraint Propagation ist ein Algorithmus, der eine vollständig konsistente Lösung konstruiert [Bar99]. Den Variablen werden Schritt für Schritt Werte zugewiesen. Das macht

er in der Art, dass jedesmal nur im Rahmen der Randbedingungen erlaubte Werte gewählt werden. Kommt dieser Prozess zu einem Ende, wird diese gültige Variablenbelegung mit der Zielfunktion bewertet. Anschließend wird die beste Lösung genommen. Gibt es keine Zielfunktion, handelt es sich um ein reines *constraint satisfaction problem* (CSP), und es wird die erste gültige Lösung zurückgegeben. Auch ist ein Mittelweg denkbar, bei dem nach einer gewissen Zeit der Lösungssuche der bis dahin am besten bewertete Lösungskandidat genommen wird. Genau das ist die grundlegende Strategie, die ich später bei der Implementation des Vorlesungsplaners (Kapitel 3) verfolge.

1.2.4.1 Knotenkonsistenz

Damit die *Constraint Propagation* die Variablen so belegt, dass alle Randbedingungen Widerspruchsfrei sind, muss sie in erster Linie in der Lage sein, dies zu Testen. Jeder Variablen kann ein Wert aus einer bestimmten Menge (*Domain*) zugewiesen werden. Um das System der Randbedingungen behandeln zu können, werden alle Bedingungen analysiert und erweitert. Jede Randbedingung die nur eine einzige Variable benützt (einwertig) wird so aufgelöst, dass aus dem Domain (D) der Variablen (V) alle unzulässigen Werte entfernt werden. Dabei werden die Variablen als Knoten und die Randbedingungen als Kanten in einem Graphen interpretiert.

Listing 1.3: Node Consistency

```
1 iteriere V aus Knotenmenge
2         iteriere X aus Domain D von V
3           wenn
4            einer der einwertigen Randbedingungen von V mit X inkonsistent ist
5           dann
6            entferne X aus dem Domain von V
```

Nun sind alle Knoten konsistent und die behandelten Randbedingungen alle erfüllt. Diese können ab nun verworfen werden.

1.2.4.2 Kantenkonsistenz

Randbedingungen die mehr als eine Variable enthalten, müssen genauer betrachtet werden. Alle, die mehr als zwei Variablen verwenden, können in ein äquivalentes System, das Hilfs- oder Ersatzvariablen einführt und nurmehr aus zweiwertigen Randbedingungen besteht ([SW99] und andere), übersetzt werden. Ziel ist es, *Kantenkonsistenz* zu erreichen.

Definition 1.2 (Kantenkonsistenz)**.**

$$\forall x \in D(V_i), \quad \exists y \in D(V_j), \quad R_{i,j}(x,y) \quad \textit{ist konsistent} \tag{1.5}$$

Diese Implikation bedeutet, dass für jeden Wert aus jedem Domain (D) einer Variablen (V_i) die Randbedingung (R) mit einer anderen Variablen (V_j) erfüllbar ist. Anzumerken ist, dass dies nicht symmetrisch ist. Ist $R_{i,j}$ konsistent, so muss dies für $R_{j,i}$ nicht gelten.

Die Algorithmen hierfür arbeiten mit Nachbarn eines Knotens bzw. im gesamten Netzwerk. Einige heißen AC-1 bis AC-3 [Mac75], AC-4 [MH86], AC-5 [Per92] [HDT92], AC-6 [BC94] und AC-7 [BFR92]. Natürlich sind das nur grundlegende Überlegungen. Konkrete Implementationen weichen davon ab, erweitern dies um zusätzliche Heuristiken, kombinieren mit anderen Verfahren oder passen sich genauer dem gestellten Problem an, als es ein so allgemeiner Zugang zulassen würde.

Der erste mögliche Schritt zum Erreichen einer Kantenkonsistenz besteht darin, eine Kante (V_i, V_j) durch Entfernen von Werten aus dem Domain von V_i konsistent zu machen. Folgender Algorithmus 1.4 macht genau das:

Listing 1.4: REVISE

```
1 REVISE(V_i,V_j)
2         DEL <- falsch
3         iteriere X aus D(V_i)
4                 gibt es kein Y aus D(V_j) damit R_i,j(X,Y) wahr ist
5                 so entferne X aus D(V_i) und DEL <- wahr
6 REVISE(V_i,V_j) <- DEL
```

Der Rückgabewert gibt an, ob ein Wert aus der Menge zulässiger Werte von V_i entfernt wurde.

Damit *alle* Kanten konsistent werden, reicht es jedoch nicht, diesen Algorithmus für alle Kanten durchlaufen zu lassen. Der Grund ist der, dass eine konsistente Randbedingung $R_{i,j}$ nach Durchlauf von Algorithmus 1.4 für $R_{k,j}$ Werte aus dem Domain von V_j entfernt haben könnte, die vorher für die Konsistenz von $R_{i,j}$ notwendig waren. Das bedeutet, dass Algorithmus 1.4 erneut für $R_{i,j}$ aufgerufen werden muss. Genau dies macht `AC-1`:

Listing 1.5: AC-1

```
1 K <- Menge aller Kanten (V_i,V_j) des Graphen
2 iteriere
3         ENDE <- falsch
4         iteriere (V_i,V_j) aus K
5                 ENDE <- REVISE(V_i,V_j) oder ENDE
6 solange bis ENDE == falsch
```

Es wird so lange Algorithmus 1.4 aufgerufen, bis es an jeder Kante keine Aktivität mehr gibt. Klarerweise ist dies sehr aufwändig, da nur eine einzige Änderung den gesamten Prozess wiederholt. Geschickter ist es, nur auf relevante Teile des Graphen einzugehen. `AC-3` – Algorithmus 1.6 – macht dies so:

Listing 1.6: AC-3

```
1 K <- Menge aller Kanten (V_i,V_j) des Graphen
2 iteriere solange K nicht leer ist
3         waehle und entferne eine Kante (V_u,V_v) aus K
4         ist
```

```
5                 REVISE(V_u,V_v) == wahr
6         dann
7                 K <- K vereinigt mit
8                 {allen Kanten (V_i,V_u), die gemeinsame Variablen teilen}
```

Es gibt hier noch mehr Erweiterungen, zum Beispiel nicht alle Werte zu testen, sondern nur einige der einzelnen Domains; das ändert jedoch nichts daran, dass reine Kantenkonsistenz noch nicht die Lösung des Problems liefert.

Das verdeutlicht folgendes Beispiel, das nur die Randbedingung `(C3)` aus dem Beispielproblem 1.2 benützt. Da wir vorausgesetzt haben, nur zweiwertige Randbedingungen zu verwenden, werden sie hier abermals so wie in 1.3 umgewandelt. Sind die zulässigen Mengen für jede der drei Variablen A, B und C jeweils $\{1, 2\}$, so macht Algorithmus 1.4 nichts, da für jedes Element des Domains jeder Variable für jede Randbedingung ein Element in einer der beiden anderen Variablen gefunden werden kann, damit die Bedingung erfüllt ist.

Damit ist hier die 2-Kantenkonsistenz erfüllt. Es ist hingegen ebenfalls klar, dass es keine gültige Belegung für alle drei Variablen geben kann. Daher müssen noch weitere Techniken eingeführt werden.

Eine Idee wäre, die Kantenkonsistenz (Definition 1.2) von zwei benachbarten Knoten auf drei oder mehr zu erweitern (3- oder k-Kantenkonsistenz). Dies ist in der Praxis jedoch nicht immer durchführbar, da der dafür notwendige Rechen- und Speicheraufwand exponentiell steigt und zu groß wird.

k-Konsistenz lässt sich so definieren:

Definition 1.3 (k-Kantenkonsistenz)**.** *Seien die Variablen $X_1, ..., X_{k-1}$ aus der Menge der Variablen X mit deren jeweiligen Grundbereichen $D_1, ..., D_{k-1}$ gegeben. Weiters haben diese die Werte $x_1, ..., x_{k-1} \in D_1, ..., D_{k-1}$, sodass alle Randbedingungen zwischen den Variablen $X_1, ..., X_{k-1}$ erfüllt sind.*

Existiert nun für eine weitere Variable X_k ein Wert x_k aus deren Domain D_k, sodass alle Randbedingungen zwischen diesen k Variablen erfüllt sind, dann spricht man von k-Kantenkonsistenz.

Praktikabler ist es, Variablen gezielt einzelne Werte aus der Menge der Zulässigen zuzuweisen und diese erneute Verkleinerung der Menge wieder im Graphen ähnlich wie Algorithmus 1.4 fortzupflanzen. Genau das ist namensgebend für die *Constraint Propagation*.

1.2.4.2.1 Beispiel Um zu zeigen wie nützlich 2-Kantenkonsistenz ist, wenden wir `AC-3` auf Beispiel 1.2 an.

- Zuerst haben Variable A, B und C den Domain $\{1, 2, 3, 4\}$.
- Knotenkonsistenz besagt für B, dass es nicht 1 oder 2 sein darf.
- Für A gilt `(C2)`, was für $D(A)$ bedeutet, dass 1, 2 und 3 nicht möglich sind.
- Umgekehrt kann B auch nicht 4 sein.

- Schließlich besagt (C3) für C, dass $D(C)$ nicht 3 oder 4 sein kann.
- Das ergibt folgende Domains: $D(A) = \{4\}$, $D(B) = \{3\}$ und $D(C) = \{1, 2\}$.

Im Vergleich zu den bisherigen, mehr auf Zufall und lokaler Suche basierenden Verfahren, ist dies ein respektables Zwischenergebnis.

Wird die Menge der Zeiten von 4 auf 5 erweitert, so sieht man gleich ein weitaus schlechteres Bild. Für $D = \{1, 2, 3, 4, 5\}$ ist $D(A) = \{4, 5\}$, $D(B) = \{3, 4, 5\}$ und $D(C) = \{1, 2, 3, 4, 5\}$.

Nun noch mehr zu den Suchmethoden, die aus diesem Zwischenergebnis tatsächliche Lösungen ermitteln.

1.2.4.3 Suchmethoden

Im Gegensatz zu Simulated Annealing (1.2.2) und Tabusuche (1.2.3), welche von einer vollständigen, aber falschen, Belegung der Variablen ausgehen und diese zu „reparieren“ versuchen, wird hier eine Lösung Schritt für Schritt aufgebaut. Das heißt, die Variablen werden in einer bestimmten Reihenfolge (siehe 1.2.6.1 auf Seite 23) abgearbeitet, indem ihnen Werte zugewiesen werden. Die dabei auftretenden Teillösungen sind mit den Randbedingungen konsistent und werden Schritt für Schritt vergrößert. Tritt im Laufe der Zuweisungen ein Problem auf – das heißt, es kann für die nächste Variable kein möglicher Wert mehr gefunden werden, sodass die neue Teillösung wieder korrekt ist – muss die Teillösung verkleinert und die letzte Zuweisung geändert werden.

Das führt unmittelbar zur Methode des *backtracking*.

1.2.4.3.1 backtracking Es werden der Reihe nach den Variablen Werte aus deren Domains zugewiesen. Variablen, denen schon ein Wert zugewiesen wurde, haben die Domaingröße 1. Muss während der Konsistenzprüfung dieser letzte Werte entfernt werden, ist der Domain dieser Variablen leer. Das bedeutet, es konnte keine gültige Belegung gefunden werden. In diesem Fall geht der Algorithmus zurück und versucht durch eine Änderung der Variablenbelegung mehr Erfolg zu haben. Er geht dabei so lange zurück, bis er eine frühere Variable findet, wo es noch freie Möglichkeiten zur Zuweisung gibt.

Algorithmus 1.7 gibt eine grobe Übersicht. Die Variablen werden einer vorher festgelegten Reihenfolge nach abgearbeitet (Zeile 1). Innerhalb dieser Schleife wird der aktuellen Variable jeder Wert zugewiesen und getestet, ob es mit der bis dahin gefundenen partiellen Lösung Inkonsistenzen gibt (Zeile 10). Trifft dies zu, so wird der nächste Wert genommen. Gibt es keinen Wert mehr, wird $D(V_i)$ wiederhergestellt und die vorhergehende Variable behandelt. Gibt es keine Widersprüche, wird mit der nächsten Variablen weitergemacht.

Listing 1.7: backtracking AC-3

```
1 i <- Index erster Variable mit |D(V_i)|>1 aus fixer Liste der Variablen
2         ist     |D(V_i)| > 1
3         dann    x <- noch nicht getesteter Wert von V_i
4         sonst   stell alten Zustand wieder her und waehle vorhergehende Variable
5
```

```
6           K <- Menge aller Kanten (V_v,V_i) des Graphen mit v<i
7           konsistent <- wahr
8           iteriere solange K nicht leer ist UND konsistent
9                   teste fuer alle Kanten (V_a,V_b) aus K
10                  konsistent <- REVISE(V_a,V_b)
11          ist
12                  konsistent == falsch
13          dann
14                  brich ab und versuche naechsten Wert
15          sonst
16                  waehle naechste Variable
```

Hier eine graphische Veranschaulichung in Abbildung 1.3 des Beispiels 1.2 mit 5 Zeitfenstern. (Zur besseren Veranschaulichung wird von höheren Werten beginnend zugewiesen)

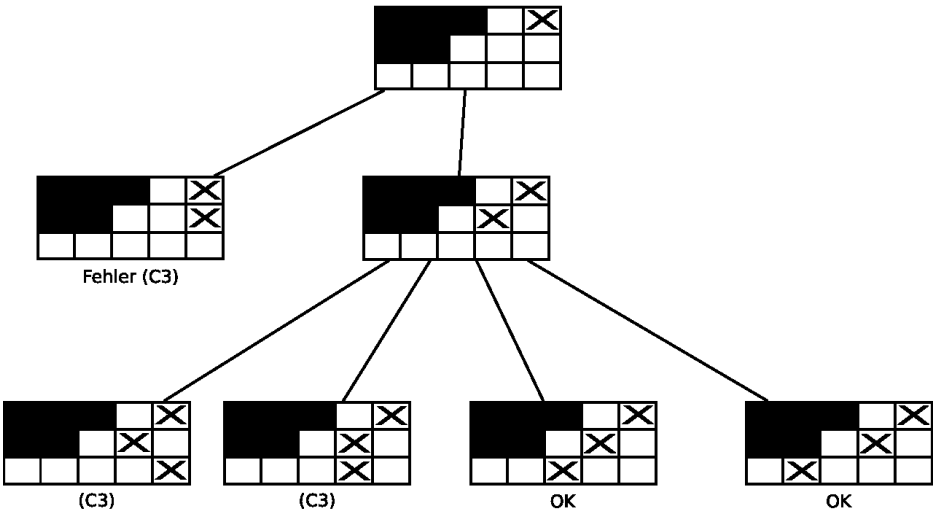


Abbildung 1.3: Suchbaum backtracking

Viele Verbesserungsmöglichkeiten sind denkbar. Der Algorithmus tendiert dazu, häufig wegen denselben falschen Zuweisungen abzubrechen. Er sollte im Lauf der Suche statt bei der letzten Variablen bei derjenigen die den Fehler verursachte neu abzweigen und – das ist die nächste Methode – er sollte im Voraus schon erkennen, welche Zuweisungen Fehler produzieren werden.

1.2.4.3.2 forward-checking *forward-checking* erweitert die Suche um den Aspekt, den Domain zukünftiger Variablen im Voraus einzuschränken. Das hat zwei unmittelbare positive Konsequenzen:

1. Wenn eine der Domains der Variablen leer wird, wird der momentane Zweig des Suchbaumes abgebrochen. Das geschieht viel früher als mit einfachem backtracking.

2. Jede Zuweisung an weitere Variablen ist immer automatisch mit den bisherigen Variablen und Randbedingungen konsistent, da inkonsistente Werte bereits im Voraus aus der Menge der zulässigen Werte entfernt wurden.

Hier ein Abriss von Algorithmus 1.8. Der Schlüssel besteht darin, die Domains der zukünftigen Variablen (vorübergehend) einzuschränken. Das schränkt die Wahlmöglichkeiten für spätere Wertzuweisungen ein (Zeile 3) und nur wenn der Domain der untersuchten zukünftigen Variablen nicht leer ist, ist die bis dahin vorgenommene Teillösung konsistent (Zeile 13).

Listing 1.8: forward-checking AC-3

```
1  i <- Index erster Variable mit |D(V_i)|>1 aus fixer Liste der Variablen
2          ist     |D(V_i)| > 1
3          dann    x <- noch nicht getesteter Wert aus D(V_i)
4          sonst   stell alten Zustand wieder her und waehle vorhergehende Variable
5
6          K <- Menge aller Kanten (V_v,V_i) des Graphen mit v>i
7          konsistent <- wahr
8          iteriere solange K nicht leer ist UND konsistent
9                  teste fuer alle Kanten (V_a,V_b) aus K
10                 wenn
11                         REVISE(V_a,V_b) == wahr
12                 dann
13                         konsistent <- |D(V_a)|>0
14         ist
15                 konsistent == falsch
16         dann
17                 brich ab und versuche naechsten Wert
18         sonst
19                 waehle naechste Variable
```

Die graphische Veranschaulichung analog zum vorhergehenden Unterpunkt zu backtracking ist hier in Abbildung 1.4 zu sehen. Die schwarzen Ellipsen symbolisieren die jeweils vorübergehend ausgeschlossenen Werte.

Im Unterschied zu vorher können Äste abgeschnitten werden und in diesem Fall direkt eine Lösung angegeben werden.

1.2.4.3.3 look-ahead Der Algorithmus *look-ahead* (auch *maintaining arc consistency* (MAC) genannt) ist abermals eine Erweiterung von *forward-checking*. Zusätzlich zu den Tests von *forward-checking* wird überprüft, ob Werte zukünftiger Variablen untereinander widersprüchlich sind. Dadurch werden noch rigoroser aussichtslose Zweige im Suchbaum abgeschnitten.

Es sollte jedoch angemerkt werden, dass dies die Arbeit erneut erhöht. Dies könnte zur Folge haben, dass der Aufwand zur Berechnung größer als notwendig werden könnte,

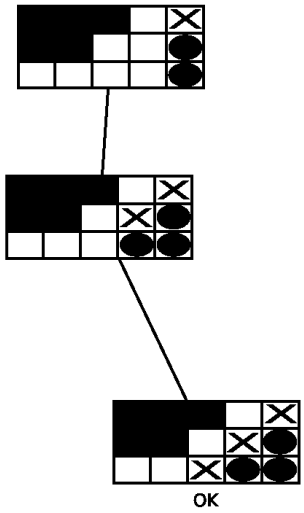


Abbildung 1.4: Suchbaum forward-checking

obwohl der Suchbaum sehr effizient durchsucht wird. Es verlagert sich lediglich der Fokus der Berechnung auf einen anderen Aspekt.

Listing 1.9: look-ahead

```
1  i <- Index erster Variable mit |D(V_i)|>1 aus fixer Liste der Variablen
2          ist     |D(V_i)| > 1
3          dann    x <- noch nicht getesteter Wert aus D(V_i)
4          sonst   stell alten Zustand wieder her und waehle vorhergehende Variable
5
6          K <- Menge aller Kanten (V_v,V_i) des Graphen mit v>i
7          konsistent <- wahr
8          iteriere solange K nicht leer ist UND konsistent
9                  teste fuer alle Kanten (V_a,V_b) aus K
10                 wenn
11                         REVISE(V_a,V_b) == wahr
12                 dann
13                         K <- K vereinigt {aller Kanten (V_m,V_n) des Graphen,
14                                 die zukuenftige Variablen mit der
15                                 momentan betrachteten Kante teilen}
16                         konsistent <- |D(V_a)|>0
17         ist
18                 konsistent == falsch
19         dann
20                 brich ab und versuche naechsten Wert
21         sonst
22                 waehle naechste Variable
```

1.2.4.4 Hilfsmittel

Die in 1.2.4.3.1 erläuterte Methode des *backtracking* hängt in der Praxis stark davon ab, in welcher Reihenfolge die Auswahl der Variablen erfolgt. In 1.2.6 werden allgemeine Heuristiken zur Auswahl der Variablen vorgestellt, jedoch gibt es in Bezug auf Constraint Propagation noch weitere Ideen [Bar99] dies zu verbessern. Es lassen sich im Graphen der Randbedingungen Strukturen finden, die sich zur geschickten Reihung der Variablen für *backtracking* nützen lassen.

Definition 1.4.

(i) Ein gerichteter Constraint-Graph ist ein linear geordneter Graph der Randbedingungen.

(ii) Die Breite eines Knotens in einem gerichteten Constraint-Graphen ist die Anzahl der Knoten, die vom vorhergehenden Knoten zu ihm hinführen.

(iii) Die Breite eines geordneten Constraint-Graphen ist das Maximum aller Knoten.

(iv) Die Breite des Constraint-Graphen ist die minimale Breite aller möglichen Ordnungen des Graphen.

Theorem 1.1. *Hat ein k-konsistenter (siehe Definition 1.3) Constraint-Graph (siehe 1.2.4.2) die Breite b mit $k > b$, so gibt es eine Suchreihenfolge ohne backtracking.*

Beweis. Es gibt einen Graphen, bei dem von jedem Knoten zum vorhergehenden maximal b Kanten weggehen. Werden die Variablen in dieser Reihenfolge initialisiert, sind sie automatisch mit allen bisherigen Wertzuweisungen konsistent da:

(i) Der Graph ist mindestens $(b + 1)$-konsistent.

(ii) Jeder Wert muss mit höchstens b anderen Variablen konsistent sein.

Folglich hat jeder baumartige (azyklische) Constaint-Graph die Breite 1, und es existiert mindestens eine Zuweisungsreihenfolge bei der bei Kantenkonsistenz (1.2.4.2) kein *backtracking* notwendig ist. □

1.2.4.4.1 cycle-cutset Die Idee ist, den Graphen der Randbedingungen in zwei Teile zu partitionieren. Ein Teil (Z) besteht aus allen zyklischen Untergraphen, und das Komplement davon (Z') ist azyklisch. Nach Theorem 1.1 kann Z' ohne *backtracking* behandelt werden.

Folgender Algorithmus 1.10 erzeugt zuerst eine Lösung für Z und erweitert diese dann ohne *backtracking* auf den gesamten Graphen.

Listing 1.10: cycle-cutset

```
1 (i)   partitioniere den Graphen in Z und Z'
2 (ii)  finde eine (weitere) Loesung fuer Z
3       gibt es keine Loesung, brich ab
4 (iii) Erzwinge gerichtete Kantenkonsistenz von Z nach Z'
```

```
5       Ist ein Domain der Variablen von Z' leer, so gehe zu (ii)
6 (iv)  Erweitere die partielle Loesung ohne backtracking auf eine
7       vollstaendige Loesung
```

Die Schwäche liegt darin, dass häufig in Schritt `(iii)` abgebrochen und zu `(ii)` zurückgekehrt werden muss.

1.2.4.4.2 MACE *MACE* erweitert die Idee von *cycle-cutset* indem er versucht Kantenkonsistenz zu erhalten, um nicht ständig abbrechen zu müssen. Dabei werden nicht alle Variablen initialisiert, sondern nur ein Teil. Dadurch gibt es weniger Tests und nicht so häufig *backtracking*.

Listing 1.11: MACE

```
1  (i)   erzwinge Kantenkonsistenz (wenn unmoeglich, brich ab)
2  (ii)  partitioniere den Graphen in Z und Z'
3        trenne Z' vom Graphen
4  (iii) fuer alle Variablen in Z
5          (a) waehle Wert fuer Z
6          (b) erzwinge Kantenkonsistenz
7              (wenn unmoeglich, gehe zu (a))
8          (c) trenne singulaere Variablen und fuege sie zu Z'
9          (d) trenne zyklusfreie Variablen und fuege sie zu Z'
10 (iv)  verbinde Variablen in Z
11       berechne gerichtete Kantenkonsistenz von Z' nach Z
12 (v)   berechne Loesung ohne backtracking in Z
```

1.2.4.4.3 zyklische Untergraphen Abschließend muss noch angemerkt werden, dass das Auffinden des Teils des Graphen für die zyklische Untermenge schwierig ist. Es gibt keinen polynomialen Algorithmus zum Auffinden des minimalen *cycle-cutset* [Bar99]. Jedoch gibt es folgende Heuristiken:

1. Sortiere die Variablen zuerst absteigend nach ihrem Grad, dann füge von der Größten beginnend alle in die Menge des *cycle-cutsets* solange der Graph zyklisch ist, hinzu.
2. Eine kleinere Menge kann erreicht werden, indem zusätzlich überprüft wird, ob die Variable Teil eines Kreises ist.
3. Berechne dynamisch für jede Variable, zu wie vielen Kreisen sie gehört und füge diejenige hinzu, welche zu den meisten gehört.

1.2.4.5 Nachbemerkung

Eine Visualisierung des Suchbaums des später behandelten Problems ist in Abbildung 3.2 auf Seite 67 zu sehen. Dort ist ansatzweise die Struktur des Suchalgorithmus erkennbar. In welcher Reihenfolge die Variablen bearbeitet werden und welche Werte ihnen zugewiesen werden ist entscheidend für den Erfolg der Suche. Näheres hierzu siehe Kapitel 1.2.6

über die Wahl der Reihenfolge der Variablen- und Wertzuweisung. Hier wurde im Rahmen der Diplomarbeit nur auf einige wenige Punkte eingegangen. Es gibt noch viele andere Methoden (siehe [Kum92]) CSP Probleme zu behandeln.

1.2.4.5.1 Überbestimmte CSPs Das von mir behandelte Problem der Vorlesungsplanung benützt sehr viele Randbedingungen. Es kann also durchaus sein, dass es keine Lösung gibt. Auch wenn es nur sehr wenige Lösungen gibt, kann es unmöglich sein eine zu finden. Daher sollten auch Lösungen in Betracht gezogen werden, die nicht alle Randbedingungen erfüllen. Das sind partielle Lösungen eines sogenannten *fuzzy constraint satisfaction problem* (FCSP) oder *weighted constraint satisfaction problem* (WCSP) (siehe [Fre89] und [Rut94]).

Definition 1.5. *Ein „fuzzy/weighted constraint satisfaction problem" besteht aus:*

- *eine Variablenmenge* $X = \{x_1, \ldots, x_n\}$,
- *einem Domain* D_i *für jede Variable (Menge möglicher Werte),*
- *einer Menge von Randbedingungen, wobei jede Randbedingung* r *mit einer Gewichtsfunktion* $g(r, w) \in [0, 1]$ *und* w*, einem Vektor von Werten, versehen ist.*

Die Lösung eines FCSPs ist eine Variablenbelegung, sodass $min\{g(r, w)\} \rightarrow max$ wird.
Die Lösung eines WCSPs ist eine Variablenbelegung, sodass $sum\{g(r, w)\} \rightarrow min$ wird.

Später implementiere ich einen Teil des gesamten Problems mit einer der FCSPs und WCSPs ähnlichen Methode, um nicht unbedingt zu erfüllende Randbedingungen behandeln zu können. Dabei gibt es wie hier eine Gewichtsfunktion und die Bewertung ist Bestandteil der Zielfunktion des CSPs (siehe 3.3.3.4.7 auf Seite 50).

Eine weitere Variante wäre, jede Randbedingung mit einer bestimmten Wahrscheinlichkeit zu aktivieren oder deaktivieren. Dadurch hat das Modell bei jeder Initialisierung andere Eigenschaften. Dies kann dazu verwendet werden, zufällige Komponenten zu behandeln.

1.2.5 Kombinierte Algorithmen

Da alle hier vorgestellten Algorithmen in unterschiedlichen Versionen mit den verschiedensten Parametern, Erweiterungen und Implementationsdetails programmiert werden können, sind sie nicht etwas starres, sondern können als Umsetzung einer Idee verstanden werden. Einige dieser Algorithmen lehnen sich an Modelle aus der Natur oder Technik an, wie zum Beispiel Simulated Annealing (siehe Kapitel 1.2.2) einem Abkühlungsprozess ähnlich ist und Genetische Algorithmen an der biologischen Evolution angelehnt sind (siehe hierzu [Ree95]). Deshalb lassen sich auf Basis dieser Algorithmen neue Ideen finden, welche für bestimmte Anwendungsfälle besser sind als jeder für sich genommen. Ein Weg dazu ist, Algorithmen zu kombinieren. Zum Beispiel wird eine Kombination aus *Constraint Propagation* und *Tabu Search* in [JL00] beschrieben.

1.2.6 Reihung der Variablen und Wertzuweisung

Bei der Constraint Propagation (1.2.4), aber auch allen anderen Suchalgorithmen bei welchen Variablen der Reihe nach mit Werten belegt werden, ist es für den Erfolg und die Geschwindigkeit der Suche entscheidend, wie das geschieht. Dabei ist einerseits die Reihenfolge der behandelten Variablen, als andererseits auch die Reihung mit welchen Werten dies passiert, entscheidend. Diese kann entweder durch eine vorher fix festgelegten Liste oder eine dynamische, dem Suchverlauf angepasste, Ordnung realisiert werden. [Smi95]

Im späteren Modell des Suchalgorithmusses wird eine dynamische Reihung der Variablen verwendet, um die Einteilung für die Vorlesungen vorzunehmen. Was hier ausgiebige praktische Tests ergaben, ist auf Seite 66 in Abschnitt 3.3.4 festgehalten.

1.2.6.1 Reihung der Variablenzuweisung

Die Zuweisung von Werten für die Variablen bedeutet, dass vorläufig aus dem Bereich der möglichen Werte einer ausgewählt wird und dieser bis auf weiteres fix ist. Die Reihenfolge dieser Zuweisung, also mit welcher Variablen begonnen wird und welche erst später behandelt werden, lässt sich entweder von Anfang an fix vorgeben, oder dynamisch im laufenden Prozess frei wählen.

Damit es eine Bewertung für die noch nicht belegten Variablen gibt, müssen zusätzliche Informationen vorhanden sein:

- die Struktur der Randbedingungen, d. h. welche Variable kommt wie oft in welchen Randbedingungen vor,
- die Menge an Werten, die die Variable noch annehmen kann (siehe 1.2.4.3.2),

  deren Größe,

  deren relative Größe, das ist das Verhältnis des gesamten Bereichs zum noch möglichen,

  die obere und untere Schranke des Domains.

Es gibt nun einige Überlegungen, wie die dynamische Reihung am besten geschehen kann:

- Wähle die Variable, die die kleinste Menge noch möglicher Werte hat. Das bedeutet, dass es im Lauf der Suche sehr rasch zu einer Situation kommen kann, wo keine Belegung mehr möglich ist. Das widerspricht zwar auf kurzer Sicht dem Wunsch, eine Lösung zu erhalten, lohnt sich aber auf längere Sicht, da aussichtslose Kombinationen schneller erkannt werden. Es verringert sich dadurch die Tiefe der Suchbäume.
- Wähle die Variable als nächstes, welche die größte Anzahl an gemeinsamen Randbedingungen mit allen schon zugewiesenen Variablen hat. Dadurch werden Konflikte mit schon zugewiesenen Variablen früher erkannt.

1.2.6.2 Reihung der Wertzuweisung

Genauso wichtig ist es, sich über die Reihenfolge der durchprobierten Werte Gedanken zu machen. Im Unterschied zur Variablenwahl verfolgt man hier den Wunsch, möglichst lange die gewählte Wertzuweisung behalten zu können. Das heißt, die Wahl des Wertes aus den noch möglichen sollte möglichst „gut“ sein.

Für das Problem der Zeiteinteilung, wie auch im stark vereinfachten Fall 1.2, zeigt sich, dass es günstig ist, bevorzugt untereinander unterschiedliche Werte zu wählen. Das erzwingt Randbedingung `(C3)`. Bei anderen Typen von Problemen könnten ganz andere Strategien nützlich sein.

Vereinheitlichen lässt sich dies so, dass die Zuweisung eines Wertes die Größen der noch möglichen Mengen der zukünftigen zu belegenden Variablen am wenigsten verkleinert. Damit wird die Anzahl an Möglichkeiten für die noch zuzuweisenden Variablen maximiert. Das kann durch das Produkt

$$\prod_{i \in D_z} \text{dsize}(\text{var}_i)$$

wobei dsize die Größe der Menge noch zuweisbarer Werte, D_z die Menge aller noch zuzuweisenden Variablen und var_i die Variablen sind, bewertet werden werden. Der Algorithmus `AC-4` [MH86] verwendet diese Idee als Test für alle Werte, was sich jedoch in einem großen Speicherplatzverbrauch niederschlägt. Verbessern lässt sich dies dadurch, nur einen oder wenige Tests vorzunehmen (`AC-6` Algorithmus, siehe [BC94]) um die quadratischen Aufwandsordnung auf eine lineare zu reduzieren.

2 Problembeschreibung

Bei dem hier behandelten Problem wird nach einem gültigen Zeitplan gesucht. Das ist eine nach Wochentagen und Stunden sortierte tabellarische Darstellung aufeinanderfolgender Blöcke von Ereignissen, die an unterschiedlichen Orten auch parallel stattfinden können. Zusätzlich müssen viele Bedingungen erfüllt werden. Konkret handelt es sich um einen Stundenplan einer Schule bzw. einem Vorlesungsplan auf einer Universität. Die beiden genannten Beispiele sind sich ähnlich und es wird auf letzteres eingegangen.

2.1 Allgemein

Für einen gültigen Vorlesungsplan einer Universität müssen verschiedenste Arten von Bedingungen erfüllt werden. Diese werden aus der noch vagen praktischen Formulierung exakt herausgearbeitet, um es mathematisch behandeln und lösen zu können. Das ist die hohe Kunst der Modellierung, und es folgen nun nebst grundlegender Feststellungen die einzelnen notwendigen Teilbedingungen, die eine akzeptable Lösung erfüllen muss.

2.2 Modell und Bedingungen

Für eine umfassende Beschreibung des Problems mit anschließender algorithmischer Behandlung werden verschiedene Objekte benötigt. Zuerst muss man sich im Klaren sein, an welchen Schrauben man zum Erreichen der Lösung drehen kann und welche unumstößlichen Tatsachen immer gelten müssen. Nach dieser Analyse sind die Verknüpfungen zwischen den einzelnen Facetten der Problemstellung bekannt, und man weiß wie sie sich untereinander verhalten.

Der entstehende Plan aller Vorlesungen besteht aus einer übersichtlich gestalteten Liste aller Vorlesungen, die jeweils zu bestimmten Zeiten beginnen und an verschiedenen Orten stattfinden. Es bietet sich daher an, die Beginnzeiten und Orte als die zu bestimmenden Variablen auszuwählen. Alternative Ansätze wären, die Belegungen der einzelnen Orte oder die Zeiteinteilungen der Vortragenden jeweils als Variable zu fixieren. Das wären andere Sichtweisen auf das Problem und während meiner Voruntersuchungen war die erstgenannte Möglichkeit diejenige, mit der sich am besten arbeiten ließ.

Eine Vorlesung kann in verschieden wählbaren Räumen zu unterschiedlichen Zeiten gehalten werden. Der Vortragende zu jeder Vorlesung ist hingegen fix zugeordnet. Da eine Vorlesung aus mehreren Teilen bestehen kann – gemeint sind Vorlesungen, welche an mehreren Tagen der Woche gehalten werden – muss zusätzlich auch jeder einzelne Teil nummeriert werden. Eine mögliche Notation der Variable (X) ist die folgende:

$$X = \{\texttt{VoID.Teil}; \texttt{Tag.Beginn}, \texttt{HSaal}\} \tag{2.1}$$

wobei

- `VoID.Teil` die Vorlesungen mit ihren Teilen indiziert. Nähere Informationen zu dieser Vorlesung werden unter diesem Schlüssel einer Tabelle entnommen.
- `Tag.Beginn` ist die Variable, die für den Zeitpunkt, an dem der Teil dieser Vorlesung eingeteilt wird, steht – also eine Kombination aus dem Wochentag und der Beginnzeit an diesem Tag.
- `HSaal` steht für den frei wählbaren Hörsaal.

Weitere Informationen zu den Vorlesungen (`V`) sind in einer anderen Tabelle verfügbar und fix vorzugeben. Aus diesen Daten werden weitere Randbedingungen hergeleitet. Ein mögliches Layout für diese Tabelle sieht so aus:

$$\begin{aligned} V = \{&\texttt{VoID.Teil}; \texttt{Titel}, \texttt{VortrID}, \texttt{Dauer}, \\ &\texttt{MaxAnzahlZuhörer}, \texttt{TypVO}, \texttt{TypZeit}, \\ &\texttt{TeileZurSelbenZeit}\} \end{aligned} \tag{2.2}$$

- `VortrID` ist ein Verweis auf den jeweils Vortragenden.
- `Dauer` gibt die Dauer dieses Teils der Vorlesung an.
- `MaxAnzahlZuhörer`ist ein Richtwert zur Wahl des Hörsaals; weil nicht jeder Hörsaal dasselbe Fassungsvermögen hat, kann es für die Auswahl von Relevanz sein kann.
- `TeileZurSelbenZeit` ist dann `WAHR`, wenn die verschiedene Teile einer Vorlesung an unterschiedlichen Tagen zur selben Zeit beginnen sollen.

Die Vortragenden (`P`) und Hörsäle (`H`) besitzen weitere Eigenschaften, die in das Modell einfließen. Es wird beispielsweise die Kapazität des Hörsaals oder die zeitliche Verfügbarkeit des Vortragenden und der Hörsäle erfasst.

$$P = \{\texttt{VortrID}; \texttt{ZeitVerfügbar}\} \tag{2.3}$$

$$H = \{\texttt{HSaal}; \texttt{AnzahlSitzplätze}, \texttt{ZeitVerfügbar}\} \tag{2.4}$$

Um eine vernünftige und für alle akzeptable Lösung zu erhalten, bedarf es zusätzlicher daraus abgeleiteter Bedingungen. Man stelle sich eine Lösung vor, bei der zeitgleich im selben Raum zwei unterschiedliche Vorlesungen geplant sind oder ein Vortragender gleichzeitig zwei Vorlesungen halten soll. Dies ist freilich inakzeptabel. Bei genauer Analyse stößt man noch auf weitere Einschränkungen, die nun vorgestellt werden. Die spätere Implementation wird an manchen Stellen etwas abweichen, da manches zwar inhaltlich

den selben Sinn hat, aber auf unterschiedliche Art eingebaut werden kann. Die technische Umsetzung der Programmierung verhält sich anders als eine auf mehr auf das Verständnis ausgelegte mathematische Erklärung.

Die folgenden Gleichungen repräsentieren die notwendigen Bedingungen des Problems. Sie sind logische Aussagen und können entweder `WAHR` oder `FALSCH` sein. Zum Lösen wird die Variable (2.1) so lange verändert, bis alle Bedingungen erfüllt sind. Es gilt, dass ein gültiger Zeitplan alle Gleichungen erfüllen muss. Sind mehrere Lösungen möglich, so wird eine Bewertungsfunktion herangezogen, die die beste Lösung bestimmt.

Zur Notation muss noch bemerkt werden, dass das Zeichen für eine `UND` Verknüpfung $\wedge$ ist, für `ODER` ein $\vee$ und eine Negation ist !. Der $\Rightarrow$ Operator ist hier die mathematische Implikation. Damit die gesamte Aussage `WAHR` ist, muss entweder das erste und das zweite Argument `WAHR` sein, oder das erste `FALSCH` und das zweite `WAHR` oder `FALSCH`. Eine mögliche Definition des $\Rightarrow$ Operators ist $A \Rightarrow B :=!(A) \vee B$. Darüber hinaus hat in meinen Formeln dieser $\Rightarrow$ gegenüber den anderen logischen Operatoren eine geringere Auswertungspriorität, um übermäßig viele Klammernebenen zu sparen. Der Punkt bedeutet, dass der hier angegebene zur Variable assoziierte Wert einer Variable gemeint ist, zum Beispiel ist mit $v.\texttt{Tag}$ der zur Vorlesung v verknüpfte Tag gemeint. Im Folgenden sei nun $x, x_1, x_2 \in \texttt{X}$ aus der Menge der Vorlesungen, wobei $x_1 \neq x_2$ ist.

Außerdem werden folgende Hilfsfunktionen verwendet, um Überschneidungen zu behandeln:

Definition 2.1.

$$x.\mathit{Ende} := x.\mathit{Beginn} + x.\mathit{Dauer} \tag{2.5}$$

$$\ddot{U}(x_1, x_2) := (x_1.\mathit{Ende} \leq x_2.\mathit{Dauer} - \mathit{Pause}) \vee (x_1.\mathit{Beginn} - \mathit{Pause} \geq x_2.\mathit{Ende}) \tag{2.6}$$

Sie bedeuten, dass die Funktion $\ddot{U}(x_1, x_2)$ dann `WAHR` ist, wenn entweder die erste Vorlesung zeitlich vor oder nach der zweiten ist und dazwischen noch die Zeitdauer `Pause` platz hat. `Pause` steht hierbei für einen globalen Parameter, der die Dauer einer zusätzlichen Zeit zwischen zwei Vorlesungen angibt.

2.2.1 Randbedingungen:

- Zwei unterschiedlichen Vorlesungen können nicht zur gleichen Zeit im selben Hörsaal stattfinden.

$$(x_1.\texttt{VoID.Teil} \neq x_2.\texttt{VoID.Teil}) \wedge (x_1.\texttt{HSaal} = x_2.\texttt{HSaal}) \Rightarrow \ddot{\texttt{U}}(x_1, x_2)$$

- Außerdem kann derselbe Vortragende zur gleichen Zeit nur eine Vorlesung halten.

$$(x_1.\texttt{VoID.Teil} \neq x_2.\texttt{VoID.Teil}) \wedge (x_1.\texttt{VortrID} = x_2.\texttt{VortrID}) \Rightarrow \ddot{\texttt{U}}(x_1, x_2)$$

- Vorlesungen desselben Typs dürfen sich nicht überlappen. Dadurch ist sichergestellt, dass es bei allen Anfängervorlesungen und weiterführenden Vorlesungen der

jeweiligen Studienzweige keine Überschneidungen gibt.

$$(x_1.\texttt{TypVO} = x_2.\texttt{TypVO}) \Rightarrow \ddot{\text{U}}(x_1, x_2)$$

Dies ist jedoch nur eine sehr vereinfachte Bedingung. In der Realität reicht es nicht, nur Vorlesungen desselben Typs zu testen, sondern es gibt weit mehr Kombinationsmöglichkeiten. So gibt es für den ersten Abschnitt Proseminare, die sich nicht mit Vorlesungen überschneiden dürfen, jedoch untereinander sehr wohl. Im zweiten Abschnitt gibt es fachspezifische Vorlesungen, die sich untereinander nicht, aber problemlos mit anderen fachspezifischen überschneiden dürfen. Weiters dürfen sich diese auch nicht mit den für alle verpflichtenden Vorlesungen überschneiden. Im Abschnitt 3.3.3.4.4 wird meine Lösung genauer vorgestellt. Es handelt sich um eine symmetrische Matrix M mit binären Einträgen, die angeben, ob sich der thematische Typ der Zeile mit dem der Spalte überschneiden darf.

$$\text{M}_{x_1.\texttt{TypVO}, x_2.\texttt{TypVO}} = 1 \Rightarrow \ddot{\text{U}}(x_1, x_2)$$

- Verschiedene Teile einer Vorlesung werden an verschiedenen Tagen vorgetragen.

$$(x_1.\texttt{VoID} = x_2.\texttt{VoID}) \wedge (x_1.\texttt{Teil} \neq x_2.\texttt{Teil}) \Rightarrow (x_1.Tag \neq x_2.Tag)$$

- Die benötigte Hörsaalkapazität muss ausreichend sein.

$$x.\texttt{MaxAnzahlZuhörer} \leq x.\texttt{HSaal.VorhandeneSitzplätze}$$

- Ein Hörsaal ist nur zu bestimmten Zeiten verfügbar. Dabei ist `ZeitVerfügbar` eine Menge von Zeitintervallen, während denen er benützt werden kann.

$$z \in x.\texttt{HSaal.ZeitVerfügbar}$$
$$\exists z : (x.\texttt{Beginn} \geq z.\texttt{Beginn}) \wedge (x.\texttt{Ende} \leq z.\texttt{Ende}) \wedge (x.\texttt{Tag} = z.\texttt{Tag})$$

- Ebenso kann ein Vortragender nur in bestimmten Zeitintervallen einen Vortrag halten.

$$z \in x.\texttt{Vortragender.ZeitVerfügbar}$$
$$\exists z : (x.\texttt{Beginn} \geq z.\texttt{Beginn}) \wedge (x.\texttt{Ende} \leq z.\texttt{Ende}) \wedge (x.\texttt{Tag} = z.\texttt{Tag})$$

- Es muss darauf geachtet werden, einige Vorlesungen nur zu bestimmten Zeiten zu halten. Zum Beispiel werden Abendvorlesungen gewöhnlich Abends gehalten. Wird kein Zeitbereich vorgegeben, so wird eine Zeitspanne von 8:00 bis 16:00 angenom-

men.

$$(x.\texttt{TypZeit} = \textit{morgens}) \Rightarrow (x.\texttt{Beginn} \geq 8:00 \wedge x.\texttt{Ende} \leq 11:00)$$
$$(x.\texttt{TypZeit} = \textit{vormittags}) \Rightarrow (x.\texttt{Beginn} \geq 10:00 \wedge x.\texttt{Ende} \leq 12:00)$$
$$(x.\texttt{TypZeit} = \textit{mittags}) \Rightarrow (x.\texttt{Beginn} \geq 11:00 \wedge x.\texttt{Ende} \leq 13:00)$$
$$(x.\texttt{TypZeit} = \textit{nachmittags}) \Rightarrow (x.\texttt{Beginn} \geq 12:00 \wedge x.\texttt{Ende} \leq 17:00)$$
$$(x.\texttt{TypZeit} = \textit{abends}) \Rightarrow (x.\texttt{Beginn} \geq 17:00 \wedge x.\texttt{Ende} \leq 20:00)$$
$$(x.\texttt{TypZeit} = \emptyset) \Rightarrow (x.\texttt{Beginn} \geq 8:00 \wedge x.\texttt{Ende} \leq 16:00)$$
$$(x.\texttt{TypZeit} = \textit{immer}) \Rightarrow (x.\texttt{Beginn} \geq 8:00 \wedge x.\texttt{Ende} \leq 20:00)$$

- Die Teile einer mehrteiligen Vorlesung sollen zur selben Zeit beginnen:

$$(x.\texttt{TeileZurSelbenZeit}) \Rightarrow (x.t_1.\texttt{Beginn} = x.t_2.\texttt{Beginn}) \quad t_1, t_2 \in \{x.\texttt{Teile}\}$$

Im endgültigen Modell wird nicht nur die Beginnzeit identisch sein, sondern auch der Hörsaal.

Das sind die Bedingungen, die jeweils nur eine Vorlesung oder die paarweisen Überschneidungsbedingungen betreffen. Im Zuge der Ausarbeitung des Modells kristallisierten sich noch weitere Typen von Bedingungen heraus, die jeweils zwei Vorlesungen betreffen. Einfaches Beispiel wäre der verpflichtende Wunsch, dass zwei unterschiedliche Vorlesungen nicht am selben Tag gehalten werden dürfen.

- Zwei Teile derselben Vorlesung oder verschiedener Vorlesungen sollen oder dürfen nicht am selben Tag, zur selben Zeit und/oder im selben Hörsaal gehalten werden.

$$(\texttt{selberTag = ja}) \Rightarrow (v_1.\texttt{Tag} = v_2.\texttt{Tag})$$
$$(\texttt{selberTag = nein}) \Rightarrow (v_1.\texttt{Tag} \neq v_2.\texttt{Tag})$$
$$(\texttt{selbeZeit = ja}) \Rightarrow (v_1.\texttt{Tag} = v_2.\texttt{Zeit})$$
$$(\texttt{selbeZeit = nein}) \Rightarrow (v_1.\texttt{Tag} \neq v_2.\texttt{Zeit})$$
$$(\texttt{selberHörsaal = ja}) \Rightarrow (v_1.\texttt{Tag} = v_2.\texttt{Hörsaal})$$
$$(\texttt{selberHörsaal = nein}) \Rightarrow (v_1.\texttt{Tag} \neq v_2.\texttt{Hörsaal})$$

- Müssen zwei Vorlesungen zeitlich unmittelbar aufeinanderfolgend gehalten werden, so kann dies so formuliert werden, wobei `aufeinanderfolgend` ein geordnetes Paar von Vorlesungen ist:

$$\forall a \in \{\texttt{aufeinanderfolgend}\}$$
$$a.v_1.\texttt{Ende} + \texttt{Pause} = a.v_2.\texttt{Beginn}$$

- Folgende Bedingung fordert, dass eine Vorlesung später als eine andere gehalten werden soll. Dies kann später am selben Tag oder an einem darauf folgenden sein.

$$\forall s \in \{\texttt{später}\}$$
$$(s.v_1.\texttt{Tag} = s.v_2.\texttt{Tag} \Rightarrow s.v_1.\texttt{Ende} < s.v_2.\texttt{Beginn}) \wedge (s.v_1.\texttt{Tag} \leq s.v_2.\texttt{Tag})$$

2.2.2 Blocklehrveranstaltungen

Manche Vorlesungen werden nicht regelmäßig über das ganze Semester angeboten, sondern in Blöcken für einige Wochen. In einem ersten Schritt zur Modellierung dieser Blocklehrveranstaltungen wird das ganze Semester, welches aus mehreren Wochen besteht, an den Stellen in einzelne Teile geteilt, wo eine der Blocklehrveranstaltungen beginnt oder aufhört. Ein Teil besteht aus einer oder mehreren Wochen. Allerdings sind dann die normalen Vorlesungen, die über das Semester in mehreren Blöcken stattfinden, zu unterschiedlichen Zeiten, weil diese Wochenblöcke unabhängig voneinander berechnet werden. Da das unerwünscht ist, muss diese blockübergreifende Bedingung erfüllt sein:

Die normalen Lehrveranstaltungen beginnen in jedem Block am selben Tag zur selben Zeit.

$$b_1, b_2 \in \{\texttt{Blöcke}\}, x_n \in \{\texttt{normale Vorlesung}\}$$
$$\forall b_1, b_2 : ((x_n)_{b_1}.\texttt{Tag} = (x_n)_{b_2}.\texttt{Tag}) \wedge ((x_n)_{b_1}.\texttt{Beginn} = (x_n)_{b_2}.\texttt{Beginn}) \wedge$$
$$\wedge (x_n)_{b_1}.\texttt{HSaal} = (x_n)_{b_2}.\texttt{HSaal}$$

Im Lauf der Entwicklung einer Implementation dieser Idee kam ich zu der Schlussfolgerung, dass dieser Ansatz zwar so weit in Ordnung ist, jedoch praktisch nicht umsetzbar. Der neue Ansatz besteht darin, die Trennung der einzelnen Blöcke aufzuheben. Das bedeutet konkret, die Lehrveranstaltungen nur einmal mit einer einzigen Variable zu versehen, auch wenn sie über mehrere Blöcke gehen. Die Überschneidungsbedingung 2.1 regelt jedoch nur dann die Zeiteinteilung, wenn sich die zu den Vorlesungen betrachteten Blöcke auch wirklich überschneiden. Diese Idee reduziert den organisatorischen Aufwand der Implementation gewaltig, es werden viel weniger Variablen benötigt (nurmehr exakt so viele wie es auch Vorlesungen gibt) und deshalb werden weniger Bedingungen aufgestellt.

Es bleibt noch, sich um die Einteilung der Vorlesungen in ihre Blöcke zu kümmern. Dies ist hier nur angedeutet und wird so gemacht, dass zu jeder Vorlesung eine Liste von Blöcken angegeben wird, an denen sie stattfindet. Ist die Durchschnittsmenge der Blockmengen nicht leer, finden diese Vorlesungen irgendwann in einem identischen Block statt. In diesem Fall dürfen sie sich nicht überschneiden.

$$\forall v1, v2 \in \{\texttt{Vorlesungen}\}, v_1 \neq v_2 :$$
$$((v_1.\texttt{Blöcke} \cap v_2.\texttt{Blöcke}) \neq \emptyset) \wedge (v_1.\texttt{Tag} = v_2.\texttt{Tag}) \Rightarrow \ddot{\text{U}}(v_1, v_2)$$

2.3 Ziel

Zu einer vollständigen Modellierung gehört auch die Wahl einer geeigneten Zielfunktion. Zwar besteht das große Problem darin, überhaupt eine akzeptable Lösung zu erhalten – doch können innerhalb der Menge der möglichen Lösungen einige den anderen vorgezogen werden. Da es sich bei einer Zielfunktion um eine zu mini- oder maximierende Funktion handelt, kann sie andere Aufgaben als die bisherigen Randbedingungen übernehmen. Ihr Wert gibt die Qualität der gefundenen Lösung an. Somit hilft sie bessere und schlechtere Lösungen zu identifizieren.

Die Frage ist nun, welche Eigenschaften bessere von schlechteren Lösungen unterscheiden. Denkbar sind diese:

W_1: – Vorlesungen sollten ihre Beginnzeit bevorzugt zur Vollen-, Viertel-, Halben- und Dreiviertelstunde haben.

W_2: – Auch wenn es nicht explizit gewünscht wird, sollen die Teile einer mehrteiligen Vorlesung zur selben Zeit stattfinden.

W_3: – Vorlesungen sollten generell zu eher früheren Tageszeiten gehalten werden.

W_4: – Die Wünsche der Vortragenden, bestimmte Zeiten oder Hörsäle zu bevorzugen oder zu meiden.

Es bietet sich an, diese Wünsche als gewichtete Summe darzustellen. Das ist die Zielfunktion.

$$Z = w_1W_1 + w_2W_2 + w_3W_3 + w_4W_4 \qquad w_i \in [0, 1] \tag{2.7}$$

Die Funktionen W_i werden dabei wie folgt berechnet:

$$W_1 := \frac{v.\texttt{Begin.Minute} \mod 15}{15} \qquad \forall v \in \texttt{Vorlesungen} \tag{2.8}$$

$$W_2 := \frac{v.t_1.\texttt{Beginn} - v.t_2.\texttt{Beginn}}{60} + (v.t_1.\texttt{HSaal} - v.t_2.\texttt{HSaal}) \qquad \forall v \in \texttt{Vorlesungen} \wedge v.\texttt{Teile} \geq 2 \tag{2.9}$$

$$W_3 := \frac{v.\texttt{Begin}}{180} \qquad \forall v \in \texttt{Vorlesungen} \tag{2.10}$$

$$
\begin{aligned}
W_4 := &+ \sum_{h \in v.\texttt{HSääle}} h \in v.\texttt{gewünschte Hörsäle} \\
&- \sum_{h \in v.\texttt{HSääle}} h \in v.\texttt{ungewünschte Hörsäle} \\
&+ \sum_{h \in v.\texttt{Vorlesungen}} [v.\texttt{Begin}; v.\texttt{Ende}] \in v.\texttt{gewünschte Zeiten} \\
&- \sum_{h \in v.\texttt{Vorlesungen}} [v.\texttt{Begin}; v.\texttt{Ende}] \in v.\texttt{unerwünschte Zeiten} \\
&\forall v \in \texttt{Vortragende}
\end{aligned}
\tag{2.11}
$$

Die Divisionen durch 60 bzw. 180 sind rein heuristisch gewählte Skalierungsfaktoren. Diese Wünsche sind außerdem nur als Ideen zu verstehen, welche in der späteren praktischen Implementation anders umgesetzt werden. Vor allem der vierte Wunsch wird eine entscheidende Rolle übernehmen.

Auf die genauen Auswirkungen dieser Bedingungen wird in Kapitel 4 eingegangen. Dort wird diskutiert, welche anderen Bedingungen auf indirekte Weise zusätzlich behandelt werden können und welche Bedeutungen sie haben.

2.4 Ein erläuterndes Beispiel

Ein kleines Beispiel ist geeignet, die Randbedingungen aus 2.2.1 zu motivieren. Freilich muss es so gewählt werden, dass möglichst viele Bedingungen zum Tragen kommen.

VoID.Teil	Titel	VortrID	Dauer	MAZ	MAT	TypVO	TypZeit
1.1	Anf1	Prof 1	2	100	2	Anf.	morgens
1.2	Anf1	Prof 1	2	100	2	Anf.	morgens
2.1	Anf2	Prof 2	2	100	2	Anf.	morgens
2.2	Anf2	Prof 2	2	100	2	Anf.	morgens
3.1	Fort1	Prof 3	3	20	2	Fort.	∅
4.1	Fort2	Prof 2	2	∅	3	Fort.	∅
4.2	Fort2	Prof 2	2	∅	3	Fort.	∅
5.1	AllgVO	Prof 3	2	∅	2	∅	∅
5.2	AllgVO	Prof 3	2	∅	2	∅	∅
6.1	PSFort	Prof 2	1	∅	2	Fort.	∅
7.1	PSAnf	Prof 1	2	100	2	Anf.	∅
8.1	VA	Prof 2	2	30	2	∅	abends
8.2	VA	Prof 2	2	30	2	∅	abends

Tabelle 2.1: Vorlesungen

Dabei stehen in Tabelle 2.1 die Abkürzungen „MAZ“ für die maximale Anzahl an Zuhörern, „MAT“ für die minimale Anzahl von Tafeln und die Einheit der „Dauer“ ist eine akademische Stunde (45 Minuten).

Die Zeitspannen aus Tabelle 2.2, in denen Hörsäle verfügbar sind, können pro Tag auch aus mehreren zusammenhängenden Blöcken bestehen. Das ist hier nicht vorgesehen, aber sollte für den allgemeinen Fall bedacht werden.

	ZeitVerfügbar			
	Tag 1	Tag 2	AnzahlTafeln	MaxAnzahlZuhörer
HSaal 1	8:00 - 20:00	8:00 - 20:00	2	200
HSaal 2	8:00 - 12:00	∅	2	200
HSaal 3	∅	12:00 - 20:00	3	50

Tabelle 2.2: Hörsäle

Ebenso gilt für die Vortragenden aus Tabelle 2.3, dass pro Tag auch mehr als eine zusammenhängende Zeitspanne vorgegeben sein könnte. Ein mögliches Beispiel wäre eine fixe Zeitspanne für eine Mittagspause.

	ZeitVerfügbar	
	Tag 1	Tag 2
Prof 1	8:00 - 12:00	8:00 - 12:00
Prof 2	8:00 - 20:00	8:00 - 20:00
Prof 3	10:00 - 15:00	10:00 - 18:00

Tabelle 2.3: Vortragende

	[min]
Pause 1	10
Pause 2	15

Tabelle 2.4: Pausen

	Tag 1			Tag 2		
	HS 1	HS 2	HS 3	HS 1	HS 2	HS 3
8:00	Anf 1		X	Anf 1	X	X
9:40	Anf 2		X	Anf 2	X	X
			X		X	X
12:00	Allg VO	X	Fort 2	Allg VO	X	Fort 2
13:30		X	PS Fort		X	Fort 1
14:00		X		PS Anf	X	
		X			X	
		X			X	
18:00	VA	X		VA	X	

Tabelle 2.5: mögliche Anordnung

Ziel des Algorithmus ist es, die 7 Vorlesungen aus Tabelle 2.1 passend anzuordnen. Dabei müssen die verfügbaren Zeiten der Hörsäle in Tabelle 2.2 und Vortragenden in Tabelle 2.3 sowie alle Randbedingungen aus Abschnitt 2.2.1 zusammen mit den Parametern für die Pausen in Tabelle 2.4 berücksichtigt werden. Tabelle 2.5 zeigt eine mögliche Lösung.

Um das zu überprüfen, müssen alle Bedingungen aus Abschnitt 2.2.1 erfüllt sein:

- Keine zwei Vorlesungen gleichzeitig im selben Raum: `WAHR`
- Jeder Vortragende hält nur eine Vorlesung zur selben Zeit: `WAHR`
- Vorlesungen desselben Typs dürfen sich nicht überlappen: `WAHR`
- Verschiedene Teile einer Vorlesung werden an verschiedenen Tagen vorgetragen: `WAHR`
- Es müssen genügend Tafeln vorhanden sein: `WAHR`
- Die benötigte Hörsaalkapazität muss ausreichend sein: `WAHR`
- Die Hörsäle sind nur zu bestimmten Zeiten verfügbar: `WAHR`
- Ein Vortragender kann nur in bestimmten Zeitintervallen einen Vortrag halten: `WAHR`
- Einige Vorlesungen werden nur zu bestimmten Zeiten gehalten: `WAHR`
- Wenn gewünscht, beginnen Teile einer mehrteiligen Vorlesung zur selben Zeit: `WAHR`

Dies konnte hier problemlos erreicht werden, jedoch ist schnell klar, wie komplex diese Aufgabe mit zunehmender Anzahl von Vorlesungen, Vortragenden, Hörsälen und Tagen wird.

3 Umsetzung mit ILOG/OPL

Der kommerzielle ILOG-Solver [ilo] ist ein aus mehreren Modulen bestehendes Programmpaket. Es gibt Lösungsprogramme für mathematische Modelle von Problemen, Bibliotheken um den Solver ansprechen zu können, Visualisierungen, Schnittstellen und „OPL“ – eine eigene Programmiersprache zum Entwerfen von mathematischen Modellen. Ziel dieses Paketes ist das Lösen von mathematischen und industriellen Problemen wie beispielsweise das Austesten kombinatorischer Hypothesen, die Routenplanung von Transportfirmen, die Berechnung der Frequenzvergabe für Mobilfunkanlagen (Erweiterung des altbekannten „Vier-Farben“-Problems), die Planung und Koordination eines Bauprojektes, das Auffinden von optimalen Einstellungen einer Maschine oder eine effiziente Sequenzierung von Automobilen in der Endmontage.

Nach einigen weiteren Vorbemerkungen zu Solver und „OPL“ werde ich meine in Kapitel 2 beschriebene Umsetzung des Problems darlegen.

3.1 Struktur

Zunächst habe ich für die Umsetzung einige kleine Versuche unternommen, um zu testen, ob das Programmpaket auf meinem Rechner funktioniert. Es gab Inkompatibilitäten zwischen ILOG-Komponenten und meinen Compilern sowie einige nicht ausreichend dokumentierte Einstellungen. Nachdem ich dies überwinden konnte, habe ich den hier beschriebenen Weg gewählt, um das Projekt umzusetzen.

Für die Entwicklung des Modells verwendete ich die OPL-Sprache. Über ein kleines JAVA-Programm, welches eine JNI-Brücke zwischen dem Code des Modells in OPL und dem Solver benützt, war es dann möglich, den Solver innerhalb eines Programms anzusprechen. JNI steht hierbei für „Java Native Interface“ und ist eine standardisierte Schnittstelle, um betriebssystemspezifischen Programmcode – in diesem Fall der in C/C++ programmierte ILOG-Solver – aufzurufen.

Weiters gibt es ein OPL-Studio, welches eine IDE zu dieser OPL-Sprache ist - diese funktionierte anfangs jedoch ebenfalls nicht, ich fand aber hier die notwendigen Vorkehrungen, um es zum Laufen zu bringen. Das funktionierte im Laufe der Entwicklung relativ zuverlässig und lässt nun alle Möglichkeiten offen, ein vollständiges JAVA-Programm zu entwickeln, welches eine leicht bedienbare Ein- und Ausgabe der Daten ermöglicht. Dies ist mir schlussendlich in Form einer dynamischen Webseite gelungen.

Die dafür entscheidenden JAVA-Zeilen sehen so aus:

Listing 3.1: JAVA Code zum Aufrufen

```
1 solver.loadInterpretedModelFileAndDataFile("<Modell>.mod","<Modell-Daten>.dat",1);
2 solver.solve();
```

3.2 OPL

„OPL“ ist eine Programmiersprache, um mathematische Modelle für Optimierungsprobleme beschreiben zu können. Sie ist im Kern ähnlich wie AMPL [FGK93] aufgebaut, wurde jedoch in vielen Punkten erweitert. Deshalb handelt es sich um eine sehr umfangreiche Sprache und ist für die unterschiedlichsten Problemtypen gerüstet. Ihre Stärke liegt vor allem darin, die gegebenen Daten nach diversen Kriterien verknüpfen und verarbeiten zu können, um die Menge aller Randbedingungen zu generieren. Dies wird häufig durch eine Schleife erreicht, die durch das „`forall`“ Schlüsselwort eingeleitet wird und oftmals nach einem „:“ durch eine weitere logische Bedingung eingeschränkt wird. Außerdem lassen sich solche Schleifen verschachteln oder die Ergebnisse jedes einzelnen Eintrags mit dem Schlüsselwort „`sum`“ aufsummieren. Dadurch lassen sich auch komplexere Bedingungen mit Auswahlkriterien kompakt angeben.

Im Vergleich zu einem herkömmlichen Computerprogramm liegt der Vorteil so einer Art von Programmiersprache darin, dass ein Entwicklungszyklus – das ist die Zeit zwischen einer Änderung im Code und der anschließenden Ausführung des Programms – sehr kurz ist, da das Skript nicht kompiliert werden muss. Das ist typisch für alle Skriptsprachen. Des weiteren wird die Essenz des Problems in komprimierter Form dargestellt und kann besser überblickt werden, ohne sich mit den notwendigen Details wie Datentypen und Ablaufmanagement der herkömmlichen Programmierung beschäftigen zu müssen. Weil somit Fehler sofort bemerkt werden, lässt sich schnell und flexibel am Modell arbeiten und jederzeit der Fortschritt verfolgen.

Ein Nachteil ist, durch die Syntax und die Fähigkeiten dieser Sprache begrenzt zu sein. Auch ist man nicht an die Eigenheiten der Sprache gewöhnt, und da man die Sprache nur grob auswendig beherrscht, wird häufig das Handbuch benötigt. Das führt unter Umständen zu umständlich formulierten Codeteilen oder längeren Überlegungen, wie eine gute Umsetzung formuliert werden kann.

3.3 Beschreibung und Funktionsweise des Modells in OPL

Es werden zwei Dateien verwendet: die eine enthält die Information über das Modell, die andere die Daten. Später im Programm wird die Daten-Datei dynamisch erzeugt und ist von der Logik des Modells unabhängig. Außerdem gibt es noch die Möglichkeit, das Modell in eine vorkompilierte Version umzuwandeln. Dies hat aber nur den Sinn, den Aufbau des Modells gegenüber Dritten geheim zu halten und dient nicht der Geschwindigkeitssteigerung.

3.3.1 Beschreibung der Daten

Ich habe die Strukturierung der Daten relativ einfach gehalten. Es müssen die Eigenschaften der Vorlesungen, der Vortragenden und der Hörsäle definiert werden. Außerdem gibt es noch eine symmetrische Matrix, die festlegt, welche Typen von Vorlesungen nicht gleichzeitig mit Typen anderer stattfinden dürfen und andere Listen, die diverse zusätzliche Bedingungen festlegen, welche im folgenden einzeln vorgestellt werden. Meine Beispieldaten sind in Anhang 7.2 auf Seite 121 zu sehen.

Damit der ILOG-Solver die Daten korrekt interpretieren kann, müssen die entsprechenden Strukturen festgelegt werden. Dies steht in der Datei für das Modell und sieht zum Beispiel so aus:

Listing 3.2: Datenstrukturen der Daten im Modell

```
1  struct sZeit {
2          zeit_h h;
3          zeit_m m;
4  };
5
6  struct sZeitSpanne {
7          Tage  tag;
8          sZeit von;
9          sZeit bis;
10 };
11
12 struct sHSaal {
13         string name;
14         int+ MaxAnzZuhoerer;
15         {sZeitSpanne} zeit;
16 };
17
18 struct sVortragender {
19         string name;
20         {sZeitSpanne} zeit;
21 };
22
23 struct sVoID {
24         string name;
25         int+   teil;
26 };
27
28 struct sVorlesung {
29         sVoID ID;
30         string profID;
31         int+ dauer;
32         int+ MaxAnzZH;
33         TypVO typVO;
```

```
34          TypZeit typZeit;
35          Boolean PCLAB;
36  };
```

Hier wird eine Zeitstruktur `sZeit` aus Stunde (Ganzzahl von 0 bis 23) und Minute (Ganzzahl von 0 bis 59) erzeugt, welche in einer Zeitspanne für Beginn `von` und Ende `bis` verwendet wird. Diese Zeitspannenstruktur wird bei den Vortragenden und Hörsälen verwendet um ihre Verfügbarkeit anzugeben. Hervorzuheben ist noch die `sVoID`-Struktur, welche die eindeutige Identifikationen einer Vorlesungseinheit ermöglicht. Einteilige Vorlesungen bestehen nur aus einem ersten Teil, Vorlesungen, die an mehreren Tagen der Woche gehalten werden, aus mehreren aufsteigend nummerierten Teilen. Diese Identifikation wird in der Datendeklaration der Vorlesungen verwendet, um sie mit ihren charakteristischen Eigenschaften zu verknüpfen. Natürlich gibt es noch etliche weitere Strukturen, die jedoch weniger wichtig sind oder später erklärt werden.

Im Laufe der Tests und der Entwicklung des gesamten Modells mussten diese Strukturen nach und nach abgeändert werden. Am wichtigsten ist dabei die Zeitstruktur, welche schlussendlich nur mehr aus einer Ganzzahl besteht, welche für die Anzahl der Viertelstunden ab Mitternacht steht. Dadurch wird die zum Auffinden der Lösung nötige Anzahl von Zeitvariablen halbiert und der Suchraum drastisch verkleinert!

3.3.2 Informationsfluss

Da das gesamte Modell mehr als die Aneinanderreihung von Einzelteilen ist, ist es für das Verständnis zunächst notwendig, eine grobe Übersicht anzugeben. Dieses Modell verknüpft die Daten untereinander, und es ist nicht sofort einsichtig, welche Teile der Daten mit welchen anderen wie in Verbindung stehen. Es gibt am Anfang eines fiktiven Informationsflusses gewisse Festlegungen, die von keinen anderen abhängen aber viel zu der konkreten Ausprägung der restlichen Daten beisteuern. Dazu zählen zum Beispiel die Anzahl und Bezeichnung der Wochentage, die Namen der Vortragenden und die Typen der Vorlesungen.

Es folgen die davon abhängigen Strukturen wie die Hörsaal- und Vorlesungsdeklaration, sowie zusätzliche Angaben zu den Vortragenden. Davon sind wiederum Daten für die Vorlesungen abhängig, die zum Beispiel Relationen zwischen einem beliebigen Paar von Vorlesungen festlegen. Im Modell selbst werden an manchen Stellen zusätzliche Verknüpfungen automatisch berechnet, die dann unbemerkt von Außen als zusätzliche Daten für die Formulierung der Bedingungen des Lösungsprozesses zur Verfügung stehen. Danach wird entsprechend einer grob vorgegebenen Suchstrategie versucht, eine Lösung zu finden, die, wenn möglich, eine Zielfunktion maximiert. Anschließend wird die gewonnene Lösung – es wird sich um Vektoren von Zahlen handeln – wieder anhand der verwendeten Daten aufbereitet und in einer für den Menschen leicht verständlichen Form dargestellt.

3.3.3 Beschreibung des Programmcodes für das Modell

Die Randbedingungen aus Kapitel 2 sind in ihrer Umsetzung etwas anspruchsvoll. Es stellte sich nach Durchsicht aller (von mir gefundenen) Möglichkeiten heraus, dass sich mit den als ungeordnete Mengen gegebenen Daten nicht besonders elegant arbeiten lässt. Vor allem, wenn eine Verknüpfung zwischen den Daten der einzelnen Vorlesungen mit den zu den jeweiligen Vorlesungen gehörigen ganzzahlig indizierten Variablen entsteht. Es müssen daher in einem Zwischenschritt die in ungeordneten Listen gegebenen Daten in Vektoren umgewandelt werden, um sie ganzzahlig zu indizieren. Des weiteren werden die eingegebenen Daten über 40 Überprüfungen unterzogen, um Inkonsistenzen auszuschließen (zum Beispiel muss die Endzeit aller Zeitspannen immer später als deren Beginnzeit sein). Danach werden entsprechende Variablen eingeführt, ihre Wertebereiche definiert, Bedingungen festgelegt und schlussendlich gibt es sogar die Möglichkeit, bei allen nichtdeterministischen Entscheidungen die Suchstrategie zu beeinflussen. Nun folgt ein detaillierter Blick auf die einzelnen Teile des Modells mit denen die logischen Grundbedingungen in einer entsprechenden Form der OPL Sprache formuliert werden.

3.3.3.1 Umformen der Daten in Vektoren

Hier wird die unsortierte und nicht indizierte Menge der Vorlesungen in einen Vektor umgeformt:

Listing 3.3: Verarbeitung der Daten im Modell

```
37 range rVO [1..card(VOset)];
38 sVorlesung VO [rVO];
39 sVorlesung fVO = VOset.first();
40 initialize forall(i in rVO)
41   VO[i] = VOset.next(fVO,i-1);
```

Dies betrifft nicht nur die Vorlesungen, sondern auch die Menge der Vortragenden und Hörsäle. Es entsteht jeweils ein neuer Vektor mit dem jeweiligen Datentyp, welcher alle angegebenen Informationen bereitstellt. Diese heißen nun `VOB` für die Daten zu den Vorlesungen, `Prof` für die mit den Vortragenden assoziierten Informationen, und `HS` steht für die Daten der einzelnen Hörsäle.

Eventuell gibt es auch eine einfachere Methode mit diesen Daten umzugehen – ohne die Länge des Vektors zusätzlich in den Daten angeben zu müssen – jedoch ist mir diesbezüglich keine bekannt.

3.3.3.2 Konsistenzprüfung der Daten

Wie erwähnt werden die Daten Tests unterzogen, um gewisse später folgende Bedingungen eindeutig formulieren zu können. Es werden also schon im Vorfeld bestimmte Möglichkeiten ausgeschlossen, damit diese Sonderfälle bei der anschließenden Implementierung im Code des Modells nicht mehr berücksichtigt werden müssen.

Hier ein Beispiel für eine Konsistenzprüfung der Daten der Hörsäle: Es wird geprüft, ob die Zeiteinträge für den Beginn und das Ende der Zeitspannen der Hörsäle die $<$-Relation erfüllen, weil Endzeiten immer nach Beginnzeiten sind.

Listing 3.4: Bedingung an die Daten der Hörsäle

```
42 assert
43   forall (h in HSaal)
44     forall (t in Tage)
45       (h.zeit[t].von <= h.zeit[t].bis);
```

Das Schlüsselwort `assert` leitet eine solche Überprüfung ein, das `forall`-Schlüsselwort iteriert in einer Schleife über alle Elemente der Menge `HSaal`, und dadurch werden alle Zeiteinträge der Hörsäle an jedem Tag durchlaufen. Abschließend steht die für alle notwendig geltende logische Bedingung.

3.3.3.3 Deklaration der Variablen

Die Variablen sind die folgenden:

Listing 3.5: Deklaration der Variablen

```
46 range rVOB [1..card(VOsetB)];
47 var zeit_4  vo_zeit[rVOB];
48 var Tage    vo_tag[rVOB];
49 var rHSaal  vo_hs[rVOB];
```

`rVOB` – das Intervall für die Indizierung aller Vorlesungen,

`vo_zeit` – die Variable der Beginnzeit jeder Vorlesung,

`vo_tag` – beinhaltet den gewählten Wochentag (Wertebereich aus der Aufzählung `Tage`, die in den Daten festgelegt wird),

`vo_hs` – Variable für den gewählten Hörsaal.

Es erwieß sich als unpraktikabel, in Stunden und Minuten zu rechnen, da der Wertebereich der Variablen zu groß wurde. Daher werden die Viertelstunden ab Mitternacht als Variable der Beginnzeit genommen. Das halbiert die Anzahl Variablen für die Zeit und schränkt die Wahlmöglichkeiten drastisch ein. Die Multiplikation mit 15 rechnet diese ohne Umstände in Minuten um und kann so mit anderen Zeiten verglichen werden. Außerdem gibt es noch die Variable `modul`, welche die Viertelstunden abermals durch die Randbedingung `vo_zeit` mod `modul` $\equiv 0$ einschränkt.

3.3.3.4 Implementation der Randbedingungen

Hier folgt eine detaillierte Beschreibung der Randbedingungen. Dabei wird so gut wie möglich ein systematischer Aufbau verfolgt, da es teilweise Wechselwirkungen zwischen den Gruppen von Randbedingungen gibt.

3.3.3.4.1 Gleichzeitigkeit der Vorlesungen Die wichtigste, und sofort einsichtigste, verlangt, dass zur selben Zeit am selben Tag nur eine Lehrveranstaltung pro Raum stattfinden kann. Der Code beschreibt genau die Bedingung, dass alle betrachteten Vorlesungen (aufsteigend sortiert und unterschiedlich) entweder zeitliche Vorgänger oder Nachfolger sein müssen, wenn sie für den selben Hörsaal am gleichen Tag eingeteilt werden. (Die Einschränkung in Zeile 50 wird später im Abschnitt über *Blocklehrveranstaltungen* auf Seite 59 erläutert.)

Listing 3.6: Randbedingung an Zeiten der Vorlesungen

```
50 forall(ordered i, j in rVO : card(ueberschneidB[i,j]) >= 1)
51   (vo_hs[i]=vo_hs[j] & vo_tag[i]=vo_tag[j]) =>
52       (vo_zeit[i]*15 + VO[i].dauer*akaH + pause[1] <= vo_zeit[j]*15
53    \/  vo_zeit[j]*15 + VO[j].dauer*akaH + pause[1] <= vo_zeit[i]*15);
```

Die entsprechende mathematische Formulierung lautet:

$$\begin{aligned} \forall v_1, v2 \in \texttt{VO}, v_1 \leq v_2 : \quad & v_1.\texttt{HSaal} = v_2.\texttt{HSaal} \wedge v_1.\texttt{Tag} = v_2.\texttt{Tag} \Rightarrow \\ & v_1.\texttt{Zeit} \cdot 15 + v_1.\texttt{Dauer} \cdot \texttt{akaH} + \texttt{pause}_1 \leq v_2.\texttt{Zeit} \cdot 15 \\ \vee \quad & v_2.\texttt{Zeit} \cdot 15 + v_2.\texttt{Dauer} \cdot \texttt{akaH} + \texttt{pause}_1 \leq v_1.\texttt{Zeit} \cdot 15. \end{aligned}$$

Diese Formeln haben den Zweck, die Syntax der OPL Sprache genauer zu erklären. Die Syntax dieser und aller folgenden Formeln ist so zu verstehen, dass `VO`, `HSaal`, `Prof` und ähnliche Mengen sind, aus denen ausgewählt wird. Diese stehen für die Menge an Vorlesungen, Hörsälen und Vortragenden. Dabei wird ohne näher darauf einzugehen vorausgesetzt, dass die Elemente dieser Mengen eine Ordnung besitzen. Des weiteren hat jedes dieser Elemente mehrere Werte, auf die mittels eines Punktes (.) referenziert wird. Beispielsweise kann ein Element e, welches aus a und b besteht, einmal den Wert 1 mittels $e.a$ bzw. 5 mittels $e.b$ haben.

Symbole wie $\forall$ stehen für `forall`, $\wedge$ und $\vee$ für `&` und $\backslash/$, $\Rightarrow$ (die Implikation) für `=>` usw. Konstanten wie `akaH`, die für die Anzahl Minuten einer akademischen Stunde steht, bleiben unübersetzt.

Vorhergehender Sachverhalt gilt auch für die Vortragenden: Sie können zur selben Zeit am selben Tag nur eine Vorlesung halten.

Listing 3.7: Gleichzeitigkeit der Vorträge einzelner Vortragenden verhindern

```
54 forall(ordered i, j in rVOB : VOB[i].profID = VOB[j].profID
55                & VOB[i].profID.subString(0,4) <> "meta"
56                & VOB[j].profID.subString(0,4) <> "meta"
57                & card(ueberschneidB[i,j]) >= 1)
58   (vo_tag[i]=vo_tag[j]) =>
59       (vo_zeit[i]*15 + VOB[i].dauer*akaH + pause[3] <= vo_zeit[j]*15
60    \/  vo_zeit[j]*15 + VOB[j].dauer*akaH + pause[3] <= vo_zeit[i]*15);
```

Außerdem gibt es in dem Modell die Vorkehrung, gewisse Zeitspannen *innerhalb* der wählbaren Zeit des Hörsaals auszuschließen. Das wird dann benötigt, wenn eine andere nicht von den Daten erfasste Vorlesung oder Veranstaltung diesen Hörsaal blockiert, er davor und danach jedoch wieder normal verwendbar ist. Dies tritt insbesondere dann auf, wenn sich zwei oder mehrere Einrichtungen einen Hörsaal teilen.

Die Zeitspannen werden im Teil der Daten beschrieben und sehen so aus:

Listing 3.8: Daten der ausgeschlossenen Zeitspannen

```
1 hsBelegt = {
2 < "hs1" Mon 60 70 >
3 < "hs3" Die 55 66 >
4 < "hs2" Mit 50 68 >
5 };
```

Der Code im Modell lautet wie folgt:

Listing 3.9: Zeitintervalle bei Hörsälen werden ausgeschlossen

```
61 forall (v in rVOB)
62  forall (h in rHSaal)
63    forall (b in rHsBelegt : HsBelegt[b].hs = HS[h].name)
64      (vo_hs[v]=h & vo_tag[v]=HsBelegt[b].tag) =>
65 (   (vo_zeit[v]*15 + VOB[v].dauer*akaH <= HsBelegt[b].von*15)
66 \/ (vo_zeit[v]                          >= HsBelegt[b].bis));
```

Die mathematische Formulierung:

$$
\begin{aligned}
&\forall v \in \texttt{VO}, h \in \texttt{HSaal}: \\
\forall z \in \texttt{Zeitspannen von h}: \quad & v.\texttt{HSaal} = h \wedge v.\texttt{Tag} \Rightarrow \\
& v_1.\texttt{Zeit} \cdot 15 + v_1.\texttt{Dauer} \cdot \texttt{akaH} \leq z.\texttt{von} \cdot 15 \\
\vee \quad & v_2.\texttt{Zeit} \geq z.\texttt{bis} \cdot 15.
\end{aligned}
$$

Es muss hier bedacht werden, dass diese Bedingung klarerweise von der vorhergehenden Einteilung der anderen Interessenten für den Hörsaal abhängen und daher erst aufgrund dieser angegeben werden kann. Werden zwei Zuteilungen parallel berechnet, hängen sie jeweils voneinander ab, und man muss sich deshalb im Voraus auf eine zeitliche Aufteilung einigen und die gegenseitig komplementären Zeitspannen jeweils als Ausschlusszeit angeben. Alternativ müsste man alle Beteiligten zu einer gemeinsamen Berechnung zusammenfassen – auch wenn zwischen ihnen nur diese geteilten Hörsäle die einzige Gemeinsamkeit sind.

3.3.3.4.2 Mehrteilige Vorlesungen Es gibt nicht nur Vorlesungen, die einmal pro Woche stattfinden und sich periodisch wiederholen, sondern auch solche, die öfters als einmal pro Woche gehalten werden – das sind die erwähnten Mehrteiligen. Die Idee für ihre Mo-

dellierung ist einfach: Jeder Teil wird als eigenständige Vorlesung interpretiert, mit eigener Dauer (d.h. die Summe aller Teile entspricht der Stundenzahl der jeweiligen Vorlesung) und separater Definition in den Daten. Was sie allerdings verbindet ist die Eigenschaft, dass keine zwei Teile einer mehrteiligen Vorlesung am selben Tag stattfinden dürfen. Dies entspricht dem, was normalerweise der Fall ist – unterbindet allerdings solche Fälle (Proseminare), wo zwei Teile einer mehrteilige Vorlesung am selben Tag gehalten werden könnten. Dies lässt sich auf die Art modellieren, dass in einer Liste genau diese Vorlesungen angeben werden und anschließend in dieser Bedingung nicht berücksichtigt werden.

Listing 3.10: Mehrteilige Vorlesungen

```
67 forall(ordered i,j in rVOB : VOB[i].ID.name=VOB[j].ID.name
68                              & (TAAST.member(VOB[i].ID.name) = 0) )
69    vo_tag[i] <> vo_tag[j];
```

Eine Bemerkung zur OPL-Syntax: In diesen zwei Zeilen wird über alle Paare von Vorlesungen iteriert, welche aufsteigend und unterschiedlich sind. Dies wird durch das Schlüsselwort `ordered` erledigt. Nach dem `:` steht eine `&` „und"-Bedingung die zusätzlich erfüllt werden muss. In diesem Fall muss der Name beider Vorlesungen ident sein und darf nicht Element der Menge `TAAST` sein (d. h. $VOB_i \notin$ `TAAST`). Dadurch bleiben genau die relevanten Vorlesungen übrig. Die anschließende Bedingung `<>` an die Variable für den Tag bedeutet, dass sie nicht gleich sein dürfen. Das entspricht exakt dem Geforderten.

Die entsprechende mathematische Formulierung lautet:

$$\forall v_1, v_2 \in \texttt{VO}, v_1 \neq v_2 : \quad v_1.\texttt{Titel} = v_2.\texttt{Titel} \wedge v_1.\texttt{Titel} \notin \texttt{TAAST} \Rightarrow v_1.\texttt{Tag} \neq v_2.\texttt{Tag}.$$

3.3.3.4.3 Wahl des passenden Hörsaals Damit die Vorträge und Veranstaltungen in den jeweiligen Hörsälen erfolgreich und ohne Komplikationen gehalten werden können, müssen noch weitere Bedingungen erfüllt sein. Diese betreffen die Anzahl der Sitzplätze und die Unterteilung der gesamten betrachteten Menge an Hörsälen in unterschiedliche Gruppen.

Anfängervorlesungen werden besonders zahlreich besucht und können deshalb nicht in zu kleinen Hörsälen abgehalten werden. Weiters ist die Kapazität eines Hörsaals durch Sitzplätze und Brandschutzbedingungen vorgegeben und darf nicht überschritten werden. Deshalb soll für jede Wahl eines Raumes für eine bestimmte Vorlesung gelten, dass die (vermutete) maximale Anzahl an Zuhörern nicht die Kapazität des Raumes übersteigt.

Listing 3.11: Bedingungen an die Hörsäle

```
70 forall(i in rVOB)
71   VOB[i].MaxAnzZH <= HSMaxAnzZuhoerer[vo_hs[i]];
```

Dies erlaubt aber auch, Vorlesungen mit nur sehr wenigen Zuhörern ebenfalls in großen Hörsälen unterzubringen. Praktisch geschieht dies jedoch nie (oder nur wenn es keine andere Möglichkeit gibt). Daher gibt es im Modell eine Vorkehrung, die Hörsäle der Größe

nach in zwei Gruppen zu teilen: groß und klein. Dies passiert in zwei Schritten: Es wird zuerst ein Wert berechnet, der der Anzahl maximal möglicher Zuhörer, bei der diese Trennung vorgenommen werden soll, entspricht. Dann werden Vorlesungen mit weniger als dieser Anzahl von Zuhörern nur in den kleineren Hörsälen untergebracht. Größere Vorlesungen gehen sich hingegen aufgrund der vorhergehenden Bedingung sowieso nur in größeren Hörsälen aus. Dabei müssen noch die Fälle ausgeschlossen werden, bei denen die maximal angegebene Zahl von Zuhörern einer Vorlesung größer als die maximale Anzahl möglicher Zuhörer eines der kleineren Hörsäle ist, aber kleiner als der Trennwert. Diese Vorlesungen können keinem Hörsaal zugewiesen werden, weil sich diese beiden Bedingungen gegenseitig ausschließen. Daher muss zuerst ein Wert berechnet werden, der der maximalen Anzahl der Zuhörer aller kleineren Hörsäle entspricht – das ist dann der verwendete Trennwert für die Zweiteilung der Vorlesungen.

Hier in Listing 3.12 wird dieser Trennwert berechnet:

Listing 3.12: Trennwert für große und kleine Hörsäle

```
72 int+ MaxSmallHS =
73   max(h in rHSaal : HS[h].MaxAnzZuhoerer <= splitSmallLarge)
74     HS[h].MaxAnzZuhoerer;
```

Folgendes Listing 3.13 beschränkt nun die Wahl der Hörsäle für kleinere Vorlesungen auf die kleineren Hörsäle. Die boolesche Variable `trennungSL` ließe sich über die Zielfunktion maximieren; die Praxis zeigte aber, dass das sehr schlecht funktioniert, und sie wird in der Suchstrategie 3.3.4 fix auf `1` gesetzt.

Listing 3.13: Trennung in große und kleine Hörsäle

```
75 forall (v in rVOB : VOB[v].MaxAnzZH <= MaxSmallHS)
76    (trennungSL = 1) => (HSMaxAnzZuhoerer[vo_hs[v]] <= MaxSmallHS);
```

Weiters ist in der Praxis eine räumliche und thematische Gruppierung der Hörsäle notwendig, weil es thematisch unterschiedliche Vorlesungen gibt, die auch in räumlich weiter entfernten Hörsälen gehalten werden. Deswegen muss bei jeder Vorlesung angegeben werden, aus welchen überhaupt für sie möglichen Hörsälen gewählt werden kann. Andererseits wird jedem Hörsaal eine Gruppenzugehörigkeit angegeben. Kompliziert wird es nun dadurch, dass ein Hörsaal mehrfach in unterschiedlichen thematischen Kontexten verwendet werden kann und nur ein Teil eines Gebäudes für eine Klassifizierung verwendet wird, während eine andere Themengruppe aus allen Hörsälen dieses Gebäudes wählt. Deshalb erwies es sich im Laufe der Untersuchungen als günstig, sowohl jedem Hörsaal als auch jeder Vorlesung eine Liste von Themen- oder besser gesagt Ortsgruppen zuzuordnen.

Folgende Zeile in der Beschreibung der Hörsäle weist dem Hörsaal `HS3` die Gruppe `UZA2M` und `UZA2` zu.

Listing 3.14: Klassifizierung der Hörsäle

```
6 <"HS3" 70 {"UZA2M" "UZA2"} [< 32 60 > < 32 50 > < 32 60 > < 32 60 > < 60 80 >]>
```

Die Lehrveranstaltungen tragen ebenfalls diese Klassifizierungen – in diesem Fall `UZA2M` und `UZA4M`.

Listing 3.15: Klassifizierung der Vorlesungen

```
7 < < "250081" 1 > .... { "UZA2M" "UZA4M" } ...>
```

Die möglichen Wahlen der Hörsäle für diese Vorlesungen bestehen nun aus der Vereinigung der Menge aller Hörsäle, die zu einer der angegeben Gruppen gehören. Allgemein lässt sich das in einem Zwischenschritt durch eine Matrix realisieren, welche über die Vorlesungen und Hörsäle indiziert ist und deren Einträge die Durchschnittsmengen dieser Klassifizierungsgruppen sind. Ist nun die Kardinalität einer entsprechenden Durchschnittsmenge größergleich 1, so kann dieser Hörsaal gewählt werden. Der Umweg über diese Hilfsmatrix ist notwendig, da es mir aufgrund der Syntax der OPL-Sprache nicht gelang direkt die Kardinalität der Durchschnittsmenge auszuwerten. Performanceeinbußen gibt es keine, da es keinen Unterschied macht, ob da oder dort die Berechnung ausgeführt wird. Folgendes Listing 3.16 zeigt sowohl die Vorverarbeitung der Daten für die Matrix `HSGselect`, als auch die Ausschlussbedingung im Modell für die Wahl des Hörsaals.

Listing 3.16: Klassifizierung der Hörsäle

```
77 {string} HSGselect[rVOB,rHSaal];
78 initialize
79    forall(i in rVOB & j in rHSaal)
80       HSGselect[i,j] = VOB[i].HSGruppe inter HS[j].HSGruppe;
81
82 forall(v in rVOB & h in rHSaal : card(HSGselect[v,h]) = 0)
83    vo_hs[v] <> h;
```

entsprechende mathematische Formulierung:

$$\forall v \in \texttt{VO}, h \in \texttt{HSaal}: \quad \texttt{HSGselect}_{i,j} := v_i.\texttt{HSGruppe} \cap h_j.\texttt{HSGruppe}$$
$$\forall v \in \texttt{VO}, h \in \texttt{HSaal}: \quad |\texttt{HSGselect}_{v,h}| = 0 \Rightarrow v.\texttt{HSaal} \neq h.$$

Da es sich hier um unterschiedliche Standorte von Hörsälen handelt, kann es vorkommen, dass ein Vortragender zwei aufeinanderfolgende Vorlesungen in zwei unterschiedlichen Hörsälen halten muss, wobei der Standortwechsel eine gewisse Zeit benötigt. Das hat zur Folge, dass diese beiden Vorlesungen nicht unmittelbar aufeinanderfolgend gesetzt werden dürfen, sondern diese Wegzeit mitberechnet werden muss.

Dies passiert auf folgende Weise: Zuerst wird in einer quadratischen Matrix angegeben, wie lange der Weg von einer zur anderen Gruppe dauert. Sie ist über alle Gruppen indiziert und braucht nicht symmetrisch sein.

Listing 3.17: Matrix der Wegzeiten

```
8  HSGruppen = { "UZA2" "UZA2M" "UZA4M" "UZA4Sch" "UZA4" "StrdlM"};
9
10 HSortswechsel =
11 [[  0  0  5  5  5 20 ]
12  [  0  0  5  5  5 20 ]
13  [  5  5  0  5  5 20 ]
14  [  5  5  5  0  5 25 ]
15  [  5  5  5  5  0 30 ]
16  [ 30 30 25 25 30  0 ]];
```

Dann wird das Minimum über alle Wegzeiten von der einen zur anderen Gruppe der beiden Vorlesungen berechnet. Dabei muss bedacht werden, dass dies nur am selben Tag und auch nur im selben Block (siehe *Blocklehrveranstaltungen* Seite 59) notwendig ist.

Listing 3.18: Wegzeiten miteinberechnen

```
84 forall(v1,v2 in rVOB : card(ueberschneidB[v1,v2]) >= 1
85                      & VOB[v1].profID = VOB[v2].profID)
86   (vo_tag[v1] = vo_tag[v2]) =>
87   (  (vo_zeit[v1] + VOB[v1].dauer - vo_zeit[v2]
88         >= min(g1 in VOB[v1].HSGruppe & g2 in VOB[v2].HSGruppe)
89               (HSortswechsel[ord(HSGruppen,g1),ord(HSGruppen,g2)]))
90  \/  (vo_zeit[v2] + VOB[v2].dauer - vo_zeit[v1]
91         >= min(g1 in VOB[v1].HSGruppe & g2 in VOB[v2].HSGruppe)
92               (HSortswechsel[ord(HSGruppen,g1),ord(HSGruppen,g2)])));
```

Es werden alle Vorlesungen im selben Block gewählt, die vom gleichen Vortragenden gehalten werden. Anschließend muss nur dann, wenn die beiden Vorlesungen am selben Tag stattfinden, die Pause zwischen den Vorlesungen größergleich dem Minimum aller Transportzeiten der möglichen Kombination der Gruppenzugehörigkeiten sein.

$$
\begin{aligned}
\forall \quad v_1, v_2 \in \mathtt{VO}: \quad & v_1.\mathtt{ProfID} = v_2.\mathtt{ProfID} \land v_1.\mathtt{Tag} = v_2.\mathtt{Tag} \Rightarrow \\
& v_1.\mathtt{Zeit} + v_1.\mathtt{Dauer} - v_2.\mathtt{Zeit} \geq \min_{g_1,g_2} \mathtt{HSortswechsel}_{g_1,g_2} \\
\lor \quad & v_2.\mathtt{Zeit} + v_2.\mathtt{Dauer} - v_1.\mathtt{Zeit} \geq \min_{g_1,g_2} \mathtt{HSortswechsel}_{g_1,g_2} \\
& \text{wobei jeweils:} \qquad g_1 \in v_1.\mathtt{HSGruppe}, g_2 \in v_2.\mathtt{HSGruppe}.
\end{aligned}
$$

Wird eine Vorlesung von mehr als nur einer Person gehalten, muss dies ebenfalls für diese berücksichtigt werden. Der Abschnitt über gemeinschaftlich gehaltene Vorlesungen auf Seite 61 beschreibt diese Modellierung und hier die entsprechende Bedingung für die mindestens notwendige Pause zwischen den Vorlesungen:

Listing 3.19: Wegzeiten bei gemeinschaftlich gehaltenen Vorlesungen mitberechnen

```
93  forall(v1,v2 in rVOB : card(ueberschneidB[v1,v2]) >= 1
94                       & VOB[v1].profID.subString(0,4)="meta"
95                       & VOB[v2].profID.subString(0,4)="meta")
96   forall(mp1,mp2 in rMP : MP[mp1].name=VOB[v1].profID
97                         & MP[mp2].name=VOB[v2].profID
98                         & card(MPueberschneid[mp1,mp2]) >= 1)
99    (vo_tag[v1] = vo_tag[v2]) =>
100   (   (vo_zeit[v1] + VOB[v1].dauer - vo_zeit[v2]
101         >= min(g1 in VOB[v1].HSGruppe & g2 in VOB[v2].HSGruppe)
102               (HSortswechsel[ord(HSGruppen,g1),ord(HSGruppen,g2)]))
103  \/   (vo_zeit[v2] + VOB[v2].dauer - vo_zeit[v1]
104         >= min(g1 in VOB[v1].HSGruppe & g2 in VOB[v2].HSGruppe)
105               (HSortswechsel[ord(HSGruppen,g1),ord(HSGruppen,g2)])));
```

Eine Kombination von alleine und in Gruppen gehaltenen Vorlesungen habe ich zugunsten einer Reduktion der Anzahl von Randbedingungen ausgelassen. Sie ließe sich aber leicht durch eine Abwandlung von Listing 3.19 erreichen, indem mit dem Schlüsselwort `member` getestet wird, ob der Name des einen alleine Vortragenden in der Menge der Vortragenden eines „meta“-Vortragenden enthalten ist.

Durch diese zusätzliche Klassifizierung der Hörsäle in räumliche und thematische Gruppen kann dieses Modell problemlos auf mehrere Fachgruppen, die gesamte Lehreinrichtung oder gar der ganzen Stadt ausgeweitet werden. De facto werden thematisch und räumlich getrennte Fachgebiete so sehr separiert, dass es einem parallelen Lösen der Gesamtaufgabe entspricht, da zwischen den Vorlesungen keine verknüpfenden Bedingungen bleiben!

3.3.3.4.4 Thematische Überschneidungen verhindern Grob gesagt wird bis jetzt nur darauf geachtet, dass sich die Vorlesungen zeitlich ausgehen. Aus Sicht der Studenten gibt es weitere Bedingungen, die ein akzeptabler Vorlesungsplan erfüllen muss. Vorlesungen derselben Fachgruppen bzw. für denselben Jahrgang der Anfänger sollen ebenfalls nicht zur selben Zeit gesetzt werden. Drastischstes Beispiel wäre das gleichzeitige Abhalten von zwei Anfängervorlesungen für das erste Semester in unterschiedlichen Hörsälen. Technisch natürlich machbar, praktisch aber nicht erwünscht.

Alle Vorlesungen, Proseminare und sonstigen Veranstaltungen können aber nicht über einen Kamm geschert 1:1 den Jahrgängen zugeordnet werden. Zu jeder Anfängervorlesung gibt es jeweils mehrere parallel gehaltene Proseminare, welche sich untereinander problemlos überlappen können. Allerdings sollten die Proseminare zu einer Vorlesung nicht mit denen einer anderen Vorlesung überlappen, damit sie gut besucht werden können. Letzteres ist allerdings nicht unbedingt erforderlich (es sei denn, es gibt nur ein einziges Proseminar – dann kann eines einer anderen Vorlesung mit mehreren Proseminaren nicht besucht werden und dieses eine Proseminar hat aus der Sicht der Überschneidungen denselben Charakter wie eine normale Vorlesung) da auch im schlimmsten Fall immer noch beide Proseminare besucht werden können. Eventuell sind hier manuelle Nachbesserungen mit Hilfe der weiter unten beschriebenen Bedingungen zur Koppelung zweier

Vorlesungen notwendig!

Die dazugehörige Modellierung ist etwas anspruchsvoller als die vorhergehenden. In den Daten wird zuerst eine Aufzählung der Vorlesungstypen vorgenommen, also ob es eine Anfängervorlesung der ersten zwei oder zweiten zwei Semester ist, eine Fortgeschrittenenvorlesung, die für alle Studenten notwendig ist oder eine, die einem Fachgebiet zugeordnet werden kann.

Dann wird eine Matrix definiert, deren Zeilen und Spalten mit dieser Auflistung indiziert werden und deren Werte `0` oder `1` sind. `0` bedeutet, dass zwischen dem Typ der entsprechenden Zeile und dem korrespondierenden Typ der jeweiligen Spalte Überschneidungen entstehen dürfen – eine `1` verhindert so eine Überschneidung. Dadurch lässt sich der Charakter einer Typisierung sehr flexibel beschreiben. Außerdem handelt es sich um eine symmetrische Matrix, die in der Konsistenzprüfung der Daten ebenfalls überprüft wird.

Listing 3.20: Charakterisierung der Überschneidungen

```
17 TypVO = { frei, Allg, Anf1, Anf2, ueA1, ueA2,
18           FortAllg, Fort1, Fort2, Fort3, Abend};
19
20 ueberschneid =
21 [[ 0 0 0 0 0 0 0 0 0 0 0 ]
22  [ 0 1 1 1 1 1 0 0 0 0 0 ]
23  [ 0 1 1 0 1 1 0 0 0 0 0 ]
24  [ 0 1 0 1 0 1 0 0 0 0 0 ]
25  [ 0 1 1 0 0 1 0 0 0 0 0 ]
26  [ 0 1 1 1 1 0 0 0 0 0 0 ]
27  [ 0 0 0 0 0 0 1 1 1 1 0 ]
28  [ 0 0 0 0 0 0 1 1 0 0 0 ]
29  [ 0 0 0 0 0 0 1 0 1 0 0 ]
30  [ 0 0 0 0 0 0 1 0 0 1 0 ]
31  [ 0 0 0 0 0 0 0 0 0 0 1 ]];
```

In diesem Beispiel lässt sich eine Trennung zwischen Anfänger und Fortgeschrittenenvorlesungen – also erstem und zweitem Abschnitt – ausmachen. Des weiteren steht in der Diagonalen überall dort eine `0`, wo untereinander Überschneidungen auftreten dürfen.

Hier nun der Codeteil des Modells, der diese Daten verwendet.

Listing 3.21: thematische Überschneidungen werden verhindert

```
106 forall(ordered i,j in rVOB :
107          ueberschneid[ord(VOB[i].typVO), ord(VOB[j].typVO)] = 1
108        & card(ueberschneidB[i,j]) >= 1 )
109   (vo_tag[i]=vo_tag[j]) =>
110     (vo_zeit[i]*15 + VOB[i].dauer*akaH + pause[1] <= vo_zeit[j]*15
111   \/ vo_zeit[j]*15 + VOB[j].dauer*akaH + pause[1] <= vo_zeit[i]*15);
```

Es wird mit dem Schlüsselwort `ordered` wie in Listing 3.6 über alle geordneten Paare von Vorlesungen iteriert. Davon werden nur diejenigen betrachtet, welche in der Matrix

den Eintrag `1` haben. Die zweite Bedingung `card(ueberschneidB[i,j]) >= 1` gehört zur Modellierung der *Blocklehrveranstaltungen* und wird weiter unten auf Seite 59 erklärt. Wenn sie am selben Tag gehalten werden, soll nun gelten, dass sie sich nicht überlappen dürfen – das ist im Grunde wieder identisch wie in Listing 3.6. Die Bedingung mit der Überschneidungsmatrix ist der essentielle Unterschied zu der dortigen Hörsaalbedingung.

Die dazugehörige mathematische Formulierung ist ganz analog wie in den bisherigen Beispielen zu verstehen.

3.3.3.4.5 Zeitlichen Bereich der Vorlesungen einschränken Für bestimmte Vorlesungen gibt es weitere zeitliche Einschränkungen. So werden Abendvorlesungen immer abends gehalten und dadurch wird die Wahlmöglichkeit ihrer Beginnzeit auf ein kleineres Zeitintervall eingeschränkt. Allgemeiner sieht es so aus: Zuerst wird in den Daten eine Auflistung der zeitlichen Typen gemacht und ihnen anschließend Zeitintervalle zugewiesen. In der Auflistung der Informationen über die Vorlesungen steht dann entweder der jeweilige zeitliche Typ, oder wenn kein besonderer Typ vorliegt, ein Platzhalter, der der gesamten sinnvollen Zeitspanne entspricht.

Listing 3.22: thematische Überschneidungen werden verhindert

```
112 TypZeit = {jederzeit morgens vormittags mittags nachmittags abends};
113
114 TypZeitZeiten = [
115    <32 68>
116    <32 48>
117    <40 48>
118    <44 52>
119    <48 68>
120    <68 84>];
```

Im Teil des Modells wird nun gefordert, dass die Beginnzeit größergleich des jeweiligen Beginns und die Endzeit kleinergleich des jeweiligen Endes der Hörsaalzeitspanne ist.

Listing 3.23: zeitliche Einschränkung

```
121 forall(i in rVOB)
122     vo_zeit[i]                        >= TypZeitZeiten[VOB[i].typZeit].von
123   & vo_zeit[i]*15+VOB[i].dauer*akaH <= TypZeitZeiten[VOB[i].typZeit].bis*15;
```

Die mathematische Formulierung sieht folgendermaßen aus:

$$\begin{aligned} \forall v \in \texttt{VO}: \quad & v.\texttt{Zeit} && \geq \texttt{TypZeitZeiten.von}_{v.\texttt{typZeit}} \\ \wedge \quad & v.\texttt{Zeit} \cdot 15 + v.\texttt{Dauer} \cdot \texttt{akaH} && \leq \texttt{TypZeitZeiten.bis}_{v.\texttt{typZeit}} \cdot 15. \end{aligned}$$

3.3.3.4.6 Ein zusätzlicher Raum Für besondere Vorlesungen wird ein weiterer Raum gebraucht. Das ist relativ spezifisch und betrifft hier nur den Raum für Computer – das PC-Labor. Es gibt theoretisch die Möglichkeit, hier in diesem Fall für mehrere Räume eine

Modellierung zu machen. Es stellt kein Problem dar auf mehrere Räume zu erweitern, was jedoch bedeuten würde, dass es für jede solche Vorlesung eine weitere Variable für die Wahl des Raumes gibt. Um die Leistungsfähigkeit nicht weiter einzuschränken, lasse ich das deshalb aus. Genauso ließe sich ein viel allgemeinerer Fall betrachten, wo zu jeder Vorlesung eine beliebige Liste von zusätzlichen Hörsälen angegeben werden kann, von denen alle zur eingeteilten Zeit frei sein müssen. Damit ließen sich zum Beispiel mehrere Übungen, die verteilt über mehrere Labors gleichzeitig ablaufen, modellieren. Dies wird mit diesem Model ebenfalls nicht abgedeckt, ließe sich aber in diese Richtung erweitern. Die Vereinigung dieser beiden Fälle wäre eine Liste von zusätzlichen Hörsälen mit einer Zahl, die angibt, wie viele von diesen Hörsälen benützt werden. Im ersten Fall wäre sie `1` für eines von den PC-Labors – im zweiten Fall entspräche sie der Anzahl von Laborhörsälen, die pro Vorlesung angegeben werden. Um diese Verallgemeinerung modellieren zu können, müsste eine rechteckige Matrix über alle Vorlesungen und deren jeweiligen zusätzlichen Räumen angegeben werden. Ihre Einträge wären boolesche `0`/`1`-Variablen. Der Großteil aller Variablen müsste auf den Wert `0` fixiert werden, weil die dazugehörigen Räume nicht zur Auswahl stehen. Aber einige können dann auch auf `1` gesetzt werden und die Zeilensumme muss der Zahl von zu belegenden Hörsälen entsprechen. Die dazugehörige Überschneidungsbedingung wäre ebenfalls sehr komplex.

Strukturell gesehen ist die hier vorliegende einfachere Form für ein PC-Labor ähnlich zu Listing 3.21, wobei abermals der entscheidende Unterschied die Auswahl der zu vergleichenden Vorlesungen ist. Nur wenn beide Vorlesungen das PC-Labor benötigen – also eine `1` eingetragen haben, wird ihr Überlappen am selben Tag verhindert. (Die Bedingung `card(ueberschneidB[v1,v2]) >= 1` betrifft Blocklehrveranstaltungen und wird ebenfalls weiter unten erklärt)

Listing 3.24: PC-Labor als zusätzlicher Raum

```
124 forall (ordered v1,v2 in rVOB : VOB[v1].PCLAB = 1 & VOB[v2].PCLAB = 1
125                               & card(ueberschneidB[v1,v2]) >= 1 )
126  (vo_tag[v1] = vo_tag[v2]) =>
127     ((vo_zeit[v1]*15 + VOB[v1].dauer*akaH < vo_zeit[v2]*15)
128   \/ (vo_zeit[v2]*15 + VOB[v2].dauer*akaH < vo_zeit[v1]*15));
```

Die entsprechende mathematische Formulierung:

$$\begin{aligned}\forall v_1, v_2 \in \mathtt{VO}: \quad & v_1 \neq v_2 \wedge v_1.\mathtt{PCLAB} = 1 \wedge v_2.\mathtt{PCLAB} = 1: \\ & v_1.\mathtt{Tag} = v_2.\mathtt{Tag} \Rightarrow \\ & \quad v_1.\mathtt{Zeit} \cdot 15 + v_1.\mathtt{Dauer} \cdot \mathtt{akaH} < v_2.\mathtt{Zeit} \cdot 15 \\ \vee \quad & v_2.\mathtt{Zeit} \cdot 15 + v_2.\mathtt{Dauer} \cdot \mathtt{akaH} < v_1.\mathtt{Zeit} \cdot 15.\end{aligned}$$

3.3.3.4.7 Wünsche Bei vielen Vorlesungen werden Wünsche an den Tag, die Zeit oder/und den Hörsaal gestellt. Grundsätzlich gibt es hier zwei Typen: Wünsche die erfüllt werden, wenn die Vorlesungen an die angegebenen Zeiten oder Hörsäle gesetzt werden, oder solche die dann erfüllt sind, wenn die dazugehörige Vorlesung nicht zu den gegebenen Zeiten und

Orten stattfindet. Außerdem gibt es prinzipiell zwei verschiedene Arten von Wünschen: solche die unbedingt erfüllt sein müssen (diese werden in Abschnitt 3.3.3.4.8 über Pflichten und Verbote behandelt), und andere die erfüllt sein sollen aber nicht müssen. Um dies vernünftig umzusetzen, werden nicht unbedingt einzuhaltende Wünsche separat von denen, die auf jeden Fall gelten müssen, behandelt. Um erstere geht es in diesem Abschnitt.

Ersteres ist in diesem Fall eine Liste von Identifikationen für die entsprechenden Vorlesungen zusammen mit den Daten für den Tag, einer optionalen Angabe für die Zeit (wenn die genaue Zeit frei wählbar ist, bekommt die Zeit den Wert `0` – es gilt jedoch weiterhin der gewählte Zeittyp – siehe Seite 49) und separat die Wahl des Hörsaals. Außerdem gibt es eine Gewichtung, die in den Wert der Zielfunktion einfließt. Dadurch kann es wichtigere Wünsche geben, die unter Umständen mehreren anderen vorgezogen werden.

In der Praxis kann ein Wunsch so aussehen: Zuerst die Identifikation des Hörsaals, dann die Gewichtung zwischen `-1` und `1` und dann eine Liste von alternativen Möglichkeiten. Hier wird der Vorlesung mit der Identifikation `<"250103"1>` eine Liste von möglichen Zeiten am Nachmittag zugewiesen und kein bestimmter Tag, da die Binärvariable `0` ist. Die nächste Vorlesung wird für Mittwoch gewünscht. Weiters soll wenn möglich Vorlesung `<"250103"1>` in den Hörsälen `"C207"` oder `"D101"` stattfinden, was oft wegen Wünschen an die Tafelgröße oder anderer Infrastrukturkriterien geschieht.

Listing 3.25: Daten spezieller Wünsche

```
32 wuenscheZeit = {
33 < < "250103" 1 > 0.2 { < 0 Mit 60 > < 0 Mon 56 > < 0 Mon 52 > } >
34 < < "250062" 1 > 0.3 { < 1 Mit  0 > } >
35 };
36 wuenscheOrt = {
37 < < "250103" 1 > 0.2 { "C207" "D101" } >
38 };
```

Im Modell wird jedem Wunsch eine binäre Hilfsvariable zugewiesen: Ist ihr Wert `WAHR`, impliziert das den Wunsch – ist er `FALSCH`, so ist seine Gültigkeit egal. Da jeder Wunsch eine Liste von möglichen Werten, die zur Erfüllung herangezogen werden können, ist, kann in einem Wunsch eine alternative Wahl der Tage, der Zeiten bzw. der Hörsäle angegeben werden. Da Zeit und Ort separat angegeben werden, ist jede mögliche Kombination davon wählbar und daher steigt die Wahrscheinlichkeit einen Wunsch erfüllen zu können.

Nun werden im Modellteil diese Daten so verarbeitet, dass allen Vorlesungen deren Identifikation der Identifikation der Wünsche gleicht nur dann der angegebene Wert für Zeit, Tag oder Ort zugeordnet wird, wenn die Hilfsvariable `WAHR` ist und auch ein tatsächlicher Wert vergeben ist.

Folgendes Listing 3.26 zeigt dies. Die ersten zwei Blöcke stehen für positive Wünsche, zuerst Zeit und dann Ort, die letzten zwei Blöcke für negative Wünsche, die bestimmte Zeiten oder Orte ausschließen. Im Detail funktioniert es so, dass zuerst die entsprechenden Wünsche gefiltert werden, anschließend die damit assoziierte Vorlesung gesucht wird und man so eine Referenz auf die Variable erhält. Im Kern steckt eine Summe über eine

boolesche Bedingung pro alternativer Wahlmöglichkeit eines jeden Wunsches, die jeweils entweder `1` für `WAHR` oder `0` für `FALSCH` sein kann.

Im ersten Fall gilt nun, dass pro Wunsch mindestens eine der in der Liste angegebenen Möglichkeiten erfüllt sein muss, um den zeitlichen Wunsch zu erfüllen. Der entsprechende negative Wunsch im dritten Block ist im Gegensatz dazu dann erfüllt, wenn keiner der in der Liste angegebenen Werte für diese Vorlesung gewählt wird.

Die Ortswünsche sind etwas anders, weil hier sowohl der Index des Hörsaals als auch jener der Vorlesung herangezogen wird. Dann wird eine Summe über alle Hörsäle genommen, die wieder boolesche Bedingungen aufsummiert.

Listing 3.26: Wünsche

```
129 forall (w in rWZeit : WlisteZeit[w].gewicht > 0)
130  forall (v in rVOB : VOB[v].ID = WlisteZeit[w].ID)
131    sum(wt in WlisteZeit[w].wuensche) (
132          (((wt.zeit <> 0)    =>  (vo_zeit[v] = wt.zeit))
133        & ((wt.bTag =  1)    =>  (vo_tag[v]  = wt.tag)))
134                              ) >= WZeit[w];
135
136 forall (w in rWOrt : WlisteOrt[w].gewicht > 0)
137  forall (v in rVOB : VOB[v].ID = WlisteOrt[w].ID)
138    sum(h in rHSaal : WlisteOrt[w].hs.member(HS[h].name))
139       (h = vo_hs[v]) = WOrt[w];
140
141 forall (w in rWZeit : WlisteZeit[w].gewicht < 0)
142  forall (v in rVOB : VOB[v].ID = WlisteZeit[w].ID)
143    sum(wt in WlisteZeit[w].wuensche) (
144          (((wt.zeit <> 0)    =>  (vo_zeit[v] <> wt.zeit))
145        & ((wt.bTag =  1)    =>  (vo_tag[v]  <> wt.tag)))
146             ) >= WZeit[w]*card(WlisteZeit[w].wuensche);
147
148 forall (w in rWOrt : WlisteOrt[w].gewicht < 0)
149  forall (v in rVOB : VOB[v].ID = WlisteOrt[w].ID)
150    sum(h in rHSaal : WlisteOrt[w].hs.member(HS[h].name))
151       (h = vo_hs[v]) <> WOrt[w];
```

Hier die entsprechenden mathematischen Formulierungen:

$$\begin{aligned}
\forall w \in \texttt{WlisteZeit}: \quad & w.\texttt{Gewicht} > 0 \\
\forall v \in \texttt{VO}: \quad & v.\texttt{ID} = w.\texttt{ID} \\
\sum_{wt \in w.\texttt{Wuensche}} \quad & ((wt.\texttt{Zeit} \neq 0 \Rightarrow v.\texttt{Zeit} = wt.\texttt{Zeit}) \\
\wedge \quad & (wt.\texttt{bTag} = 1 \Rightarrow v.\texttt{Tag} = wt.\texttt{Tag})) \geq w.\texttt{WZeit},
\end{aligned}$$

$$\forall w \in \texttt{WlisteOrt}: \quad w.\texttt{Gewicht} > 0$$
$$\forall v \in \texttt{VO}: \quad v.\texttt{ID} = w.\texttt{ID}$$
$$\sum_{h \in \texttt{HSaal}: h.\texttt{Name} \in w.\texttt{hs}} (h = v.\texttt{HSaal}) = w.\texttt{WOrt},$$

$$\forall w \in \texttt{WlisteZeit}: \quad w.\texttt{Gewicht} < 0$$
$$\forall v \in \texttt{VO}: \quad v.\texttt{ID} = w.\texttt{ID}$$
$$\sum_{wt \in w.\texttt{Wuensche}} ((wt.\texttt{Zeit} \neq 0 \Rightarrow v.\texttt{Zeit} \neq wt.\texttt{Zeit})$$
$$\wedge \quad (wt.\texttt{bTag} = 1 \Rightarrow v.\texttt{Tag} \neq wt.\texttt{Tag})) \geq w.\texttt{WZeit} \cdot |w.\texttt{wuensche}|,$$

$$\forall w \in \texttt{WlisteOrt}: \quad w.\texttt{Gewicht} < 0$$
$$\forall v \in \texttt{VO}: \quad v.\texttt{ID} = w.\texttt{ID}$$
$$\sum_{h \in \texttt{HSaal}: h.\texttt{Name} \in w.\texttt{hs}} (h = v.\texttt{HSaal}) \neq w.\texttt{WOrt}.$$

Abschließend muss dem Solver über eine Zielfunktion mitgeteilt werden, dass er möglichst viele der Wünsche erfüllen muss. Das passiert durch das Maximieren der Summe über alle Werte der binären Hilfsvariablen multipliziert mit deren Gewichtung. Das Aufsplitten in alle vier Fälle hat den Sinn, bei Bedarf die unterschiedlichen Arten von Wünschen zusätzlich gegeneinander gewichten zu können. (Listing 3.27 zeigt einen Teil der tatsächlich verwendeten Zielfunktion)

Listing 3.27: Maximierung der erfüllbaren Wünsche

```
152 maximize
153  + sum(w in rWZeit: WlisteZeit[w].gewicht > 0) (WlisteZeit[w].gewicht * WZeit[w])
154  + sum(w in rWOrt : WlisteOrt[w].gewicht  > 0) (WlisteOrt[w].gewicht  * WOrt[w])
155  - sum(w in rWZeit: WlisteZeit[w].gewicht < 0) (WlisteZeit[w].gewicht * WZeit[w])
156  - sum(w in rWOrt : WlisteOrt[w].gewicht  < 0) (WlisteOrt[w].gewicht  * WOrt[w])
```

die mathematische Formulierung:

$$\begin{aligned}
(\quad & + \sum_{w \in \texttt{WZeit}: w.\texttt{Gewicht} > 0} (w.\texttt{Gewicht} \cdot w) \\
& + \sum_{w \in \texttt{WOrt}: w.\texttt{Gewicht} > 0} (w.\texttt{Gewicht} \cdot w) \\
& - \sum_{w \in \texttt{WZeit}: w.\texttt{Gewicht} < 0} (w.\texttt{Gewicht} \cdot w) \\
& - \sum_{w \in \texttt{WOrt}: w.\texttt{Gewicht} < 0} (w.\texttt{Gewicht} \cdot w) \quad) \rightarrow \max.
\end{aligned}$$

3.3.3.4.8 Erzwungene Zeiten und Orte – Pflichten/Verbote Zu den im vorhergehenden Absatz besprochenen Wünschen gibt es wie gesagt noch das unbedingt zu erfüllende Pen-

dant: Pflichten (und Verbote). Werden diese Wünsche nicht erfüllt, so ist das Problem nicht gelöst. Es wird eine Vorlesung mit einer bestimmten Zeit oder frei wählbar, einem bestimmten Tag oder erneut wahlweise mit einem Ort, verknüpft.

Die Daten sind folgendermaßen aufgebaut:

Listing 3.28: Pflichten

```
39 pflichten = {
40    < < "250024" 1 > 1 Mon 32 "HS1"  >
41    < < "250103" 1 > 0 Fre  0 "D101" >
42 };
```

Diese beiden Zeilen bedeuten: Es muss die erste angegebene Vorlesung am Montag um 8 Uhr im Hörsaal `HS1` gehalten werden, die zweite Vorlesung zu einer beliebigen Zeit und an einem beliebigen Tag (die `0` vor `Fre` bedeutet, dass der Wert für den Tag ignoriert werden soll) in Hörsaal `D101`.

Der Code des Modells setzt dies relativ direkt um:

Listing 3.29: Pflichten

```
157 forall (p in pflichten)
158    forall(v in rVOB : VOB[v].ID = p.ID)
159        ((p.zeit <> 0) => (vo_zeit[v] = p.zeit))
160      & ((p.bTag =  1) => (vo_tag[v]  = p.tag));
161 forall (p in pflichten)
162    forall (v in rVOB : VOB[v].ID = p.ID)
163       forall (h in rHSaal : HS[h].name = p.hs)
164          (p.hs <> "none") => (vo_hs[v] = h);
```

Sollen bestimmte Zeiten, Tage oder Orte ausgeschlossen werden, so gibt es ganz analog eine Liste von Ausschlüssen. Auf die wird hier nicht weiter eingegangen.

Die entsprechende mathematische Formulierung besteht aus einer Reihe bedingter Implikationen:

$$\begin{aligned}
\forall p \in \texttt{Pflichten}, \forall v \in \texttt{VO}: \quad & v.\texttt{ID} = p.\texttt{ID}: \\
& p.\texttt{Zeit} \neq 0 \Rightarrow v.\texttt{Zeit} = p.\texttt{Zeit} \wedge p.\texttt{bTag} = 1 \Rightarrow v.\texttt{Tag} = p.\texttt{Tag}, \\
\forall p \in \texttt{Pflichten}, \forall v \in \texttt{VO}: \quad & v.\texttt{ID} = p.\texttt{ID}, \forall h \in \texttt{HS}: h.\texttt{Name} = p.\texttt{HSaal}: \\
& p.\texttt{HSaal} \neq \text{none} \Rightarrow v.\texttt{HSaal} = h.
\end{aligned}$$

3.3.3.4.9 Voneinander unabhängige Vorlesungen koppeln Des öfteren werden Vorlesungen oder Teile von Vorlesungen untereinander gekoppelt. Die folgenden Konstruktionen verwenden ein Paar von Identifikatoren für den Bau der Randbedingungen. Die folgenden Unterkapitel zeigen die dafür im Modell zur Verfügung stehenden Möglichkeiten. Freilich

gibt es theoretisch auch die Möglichkeit drei oder mehrere Vorlesungen zu koppeln, was sich aber in der Regel in mehrere Paare zerlegen lässt. Außerdem ist zu bedenken, dass die Bedingungen, die alle Teile einer Vorlesung betreffen, bereits abgehandelt wurden und mittels solcher angegeben werden sollten.

3.3.3.4.9.1 Selbe/unterschiedliche Zeiten/Orte Gelegentlich müssen unterschiedliche Vorlesungen im selben Raum und/oder zur selben Beginnzeit und/oder am selben Tag abgehalten werden. Als Beispiel dienen abermals die Einführungsvorlesungen. Hier gibt es als Einführung eine Blocklehrveranstaltung und später im Semester eine Vorlesung, welche genau zur selben Zeit, am selben Tag und im selben Raum stattfinden soll. Es muss dabei sichergestellt sein, dass diese beiden Vorlesungen in keinem gemeinsamen Block sind, weil es sonst zu einer Kollision kommt. Dies beschreibt Zeile 44 im folgenden Listing, wobei die 1-Werte für die erzwungene Gleichheit von Tag, Beginnzeit und Hörsaal steht.

Listing 3.30: gekoppelte Vorlesungen

```
43 selbeZeit = {
44    < < "anf1  " 1 > < "einfA " 1 > 1 1 1 >
45    < < "PSAnf1" 1 > < "PSAnf2" 1 > 1 1 0 >
46 };
```

Im Kontrast dazu werden manchmal auch unterschiedliche Tage für zwei angegebene Teile von Vorlesungen gewünscht. Dies ist dann notwendig, wenn die Vorlesung und ihr dazugehöriges Proseminar oder die beiden alternativ besuchbaren zu einer Vorlesung gehörenden Proseminare nicht am selben Tag gehalten werden sollen. Letzteres wird im folgenden Listing 3.31 auf die Art beschrieben, dass der Wert für den Tag -1 ist. Es handelt sich also an dieser Stelle um eine ternäre Variable mit den Werten -1, 0 und 1, wodurch dieser und der vorhergehende Punkt gemeinsam behandelt werden können.

Listing 3.31: gekoppelte Vorlesungen

```
47 < < "250060" 1 > < "250060" 2 > -1 0 0 >
```

Der dazugehörende Code nimmt sich für jede dieser Datensatzeinträge genau die Vorlesungen vor, deren Identifikationen in den Bedingungen angegeben sind. Daraufhin erzwingt er für die mit 1 oder -1 markierten Variablen Gleichheit oder Ungleichheit. Zeitliche Ungleichheit bedeutet hier etwas mehr, nämlich nicht nur dass die Beginnzeiten unterschiedlich sind, sondern keine zeitliche Überlappung innerhalb der ganzen Woche stattfindet.

Listing 3.32: unterschiedliche Vorlesungen koppeln

```
165 forall (sz in selbeZeit)
166    forall (v1,v2 in rVOB : VOB[v1].ID = sz.ID1 & VOB[v2].ID = sz.ID2)
167      (  ((sz.tag  =  1) => (vo_tag[v1]  = vo_tag[v2]))
168       & ((sz.zeit =  1) => (vo_zeit[v1] = vo_zeit[v2]))
```

```
169       & ((sz.hs   =  1) => (vo_hs[v1]   = vo_hs[v2]))
170       & ((sz.tag  = -1) => (vo_tag[v1]  <> vo_tag[v2]))
171       & ((sz.zeit = -1) =>
172            ( vo_tag[v1] = vo_tag[v2] =>
173                 ( vo_zeit[v1]*15 + VOB[v1].dauer <= vo_zeit[v2]*15
174            \/     vo_zeit[v2]*15 + VOB[v2].dauer <= vo_zeit[v1]*15)))
175       & ((sz.hs   = -1) => (vo_hs[v1]   <> vo_hs[v2]))  );
```

Die mathematische Formulierung ist ähnlich wie bei den Pflichten und Verboten auf Seite 54 eine Reihe bedingter Implikationen:

$$
\begin{aligned}
\forall sz \in \texttt{selbeZeit}, \forall v_1, v_2 \in \texttt{VO}: \quad & v_1.\texttt{ID} = sz.\texttt{ID}_1 \wedge v_2.\texttt{ID} = sz.\texttt{ID}_2 : \\
& sz.\texttt{Tag} = 1 \Rightarrow v_1.\texttt{Tag} = v_2.\texttt{Tag} \\
\wedge \quad & sz.\texttt{Zeit} = 1 \Rightarrow v_1.\texttt{Zeit} = v_2.\texttt{Zeit} \\
\wedge \quad & sz.\texttt{HSaal} = 1 \Rightarrow v_1.\texttt{HSaal} = v_2.\texttt{HSaal} \\
\wedge \quad & sz.\texttt{Tag} = -1 \Rightarrow v_1.\texttt{Tag} \neq v_2.\texttt{Tag} \\
\wedge \quad & (sz.\texttt{Zeit} = -1 \Rightarrow \\
& \quad v_1.\texttt{Tag} = v_1.\texttt{Tag} \Rightarrow \\
& \quad (v_1.\texttt{Zeit} \cdot 15 + v_1.\texttt{Dauer} \leq v_2.\texttt{Zeit} \cdot 15 \\
& \quad \vee v_2.\texttt{Zeit} \cdot 15 + v_2.\texttt{Dauer} \leq v_1.\texttt{Zeit} \cdot 15)) \\
\wedge \quad & sz.\texttt{HSaal} = -1 \Rightarrow v_1.\texttt{HSaal} \neq v_2.\texttt{HSaal}.
\end{aligned}
$$

3.3.3.4.9.2 Teile aufsteigend Die Teile einer mehrteilige Vorlesung können vom Programm in jeder beliebigen Reihenfolge gesetzt werden. Sie müssen nur an unterschiedlichen Tagen sein. Mit der folgenden Bedingung wird ihre Reihenfolge festgelegt. Das hat dann Auswirkungen, wenn einzelne Teile mit Teilen anderer Vorlesungen verknüpft sind (zum Beispiel der erste Teil einer Vorlesung mit dem letzten einer anderen). Erst durch diese Bedingung wird der Teil mit der höchsten Teilnummer zum zeitlich letzt gereihten Teil! Dadurch scheinen die einzelnen Teile in der eingegebenen Reihenfolge wieder im Ergebnis auf. Weiters ist das Setzen dieser Bedingung für die Berechnung der Lösung sehr nützlich, wenn alle Teile einer Vorlesung die genau selbe Dauer haben: es wird die Menge aller symmetrischen Lösungen mit der jeweils exakt selben Bedeutung auf einen ausgezeichneten Fall reduziert. Der durch diese zusätzlichen Randbedingungen verkleinerte Suchraum wirkt sich positiv auf die Effizienz des Solvers aus ohne eine einzige Lösung ausschließen zu müssen.

Der Code in Listing 3.33 erzwingt dieses Verhalten für alle in der Liste `teileAufsteigend` spezifizierten Vorlesungsteile. (Es wurde bewusst auf den Fall mit gleichem Tag verzichtet.)

Listing 3.33: Teile aufsteigend sortieren

```
176 forall (ta in teileAufsteigend)
177    forall (ordered v1, v2 in rVOB :  VOB[v1].ID.name = ta
178                                   &  VOB[v2].ID.name = ta
```

```
179                                        & (VOB[v1].ID.teil < VOB[v2].ID.teil) )
180       vo_tag[v1] < vo_tag[v2];
```

Die dazugehörige mathematische Formulierung sieht folgendermaßen aus:

$$\forall ta \in \texttt{teileAufsteigend}: \quad \forall v_1, v_2 \in \texttt{VO}: \\ v_1 < v_2 \wedge v_1.\texttt{ID}_{\texttt{Name}} = ta \wedge v_2.\texttt{ID}_{\texttt{Name}} = ta \wedge v_1.\texttt{ID}_{\texttt{Teil}} < v_2.\texttt{ID}_{\texttt{Teil}}: \\ v_1.\texttt{Tag} < v_2.\texttt{Tag}.$$

3.3.3.4.9.3 Mehrteilige Vorlesungen auch am selben Tag – „ TAAST “ Eine weitere Bedingung, die sich mit den Teilen einer Vorlesung beschäftigt, ermöglicht es, Teile auch am selben Tag setzen zu können. Dies ist nicht mit der vorhergehenden Bedingung verträglich. Gebraucht wird sie vor allem für Proseminare und ist unbedingt notwendig, wenn ein Proseminar mehr Einheiten als Wochentage hat.

Im Code des Modells erledigt dies die zusätzliche Einschränkung in Zeile 182 von Listing 182, wobei das Schlüsselwort `member` testet, ob der `name` der Vorlesung in der Menge `TAAST` enthalten ist.

Listing 3.34: Teile auch am selben Tag

```
181 forall(ordered i,j in rVOB : VOB[i].ID.name=VOB[j].ID.name
182                     & (TAAST.member(VOB[i].ID.name) = 0) )
183    vo_tag[i] <> vo_tag[j];
```

die mathematische Formulierung:

$$\forall v_1, v_2 \in \texttt{VO}: \quad v_1.\texttt{ID}_{\texttt{Name}} = v_2.\texttt{ID}_{\texttt{Name}} \wedge v_1.\texttt{ID}_{\texttt{Name}} \notin \texttt{TAAST}: \\ v_1.\texttt{Tag} \neq v_2.\texttt{Tag}.$$

3.3.3.4.9.4 Zwei Vorlesungen zeitlich sortieren – „später“ Auch können zwei Teile unterschiedlicher Vorlesungen aufsteigend sortiert werden. Dies erledigt dieser Code:

Listing 3.35: Teile zeitlich aufeinanderfolgend

```
184 forall (sp in spaeter)
185    forall (v1, v2 in rVOB : VOB[v1].ID = sp.v1 & VOB[v2].ID = sp.v2)
186        ((vo_tag[v1] = vo_tag[v2]) =>
187         (vo_zeit[v1]*15 + VOB[v1].dauer <= vo_zeit[v2]))
188     &   vo_tag[v1] <= vo_tag[v2];
```

Es kann zwei Fälle geben: Entweder die beiden Teile sind am selben Tag, so muss das Ende des ersten Teils vor dem Anfang des zweiten sein – oder der eine Tag ist früher als der andere. Die `&` Verknüpfung ist deswegen notwendig, weil eine `=>`-Implikation wahr ist, wenn die Prämisse falsch ist und deswegen immer der Fall, dass die erste Vorlesung an einem späteren Tag als die zweite sein könnte ausgeschlossen werden muss.

Weiters muss darauf geachtet werden, die Teile einer mehrteiligen Vorlesungen jeweils zu sortieren, weil sonst ein nicht von dieser Bedingung erfasster Teil zeitlich vor denen der anderen Vorlesung sein kann und folglich die „später“-Bedingung zerstört wird.

Die mathematische Formulierung:

$$\begin{aligned}\forall sp \in \texttt{später}, v_1, v_2 \in \texttt{VO}: \quad & v_1.\texttt{ID} = sp.v_1 \wedge v_2.\texttt{ID} = sp.v_2 : \\ & v_1.\texttt{Tag} = v_2.\texttt{Tag} \Rightarrow v_1.\texttt{Zeit} \cdot 15 + v_1.\texttt{Dauer} \leq v_2.\texttt{Zeit} \\ \wedge \quad & v_1.\texttt{Tag} \leq v_2.\texttt{Tag}.\end{aligned}$$

3.3.3.4.9.5 Teile zur selben Zeit abhalten – „TZSZ“ Diese Bedingung erzwingt die oft übliche Konvention, die Teile einer mehrteiligen Vorlesung an den jeweiligen Tagen zur selben Zeit im selben Hörsaal zu halten. Die dazugehörige Formulierung in den Daten und im Modell gestaltet sich so:

Wenn diese Eigenschaft gewünscht wird, muss bei den Daten der Identifikationsname der Vorlesung in die Menge `TZSZ` eingetragen werden. Das erzwingt, dass alle Teile zur selben Zeit und am selben Ort gewählt werden. Dadurch bleibt bei den weiteren Teilen der Vorlesung einzig der Tag als Freiheitsgrad übrig. Letzteres hat große Auswirkungen auf den abzusuchenden Lösungsraum weil er drastisch verkleinert wird. Natürlich steigt damit auch die Wahrscheinlichkeit, unvereinbare Situationen zu schaffen, wenn keine Lösung gefunden werden kann. Dies haben diverse Tests, die zur Konstruktion dieses Modells führten, gezeigt.

Listing 3.36: Die Teile im selben Hörsaal zur gleichen Zeit

```
189 forall (tzsz in TZSZ)
190   forall (ordered v1,v2 in rVOB : VOB[v1].ID.teil=1 & VOB[v2].ID.teil>1
191             & VOB[v1].ID.name = VOB[v2].ID.name
192             & VOB[v1].ID.name = tzsz)
193     vo_zeit[v1] = vo_zeit[v2] & vo_hs[v1] = vo_hs[v2];
```

Die entsprechende mathematische Formulierung:

$$\begin{aligned}\forall t \in \texttt{TZSZ}: \quad & \forall v_1, v_2 \in \texttt{VO} : v_1 < v_2 \wedge v_1.\texttt{ID}_{\texttt{Teil}} = 1 \wedge v_2.\texttt{ID}_{\texttt{Teil}} > 1 \\ & \wedge \quad v_1.\texttt{ID}_{\texttt{Name}} = v_2.\texttt{ID}_{\texttt{Name}} \wedge v_1.\texttt{ID}_{\texttt{Name}} = t : \\ & v_1.\texttt{Zeit} = v_2.\texttt{Zeit} \wedge v_1.\texttt{HSaal} = v_2.\texttt{HSaal}.\end{aligned}$$

3.3.3.4.9.6 Unmittelbar aufeinanderfolgende Vorlesungen Eine eher selten benötigte Bedingung betrifft zwei unterschiedliche Vorlesungen, die hintereinander stattfinden sollen. Beispielsweise soll gleich anschließend an eine normale Vorlesung ein dazugehörendes Proseminar in einem Labor gehalten werden. Diese beiden Einheiten können gekoppelt werden, da sie ansonsten voneinander unabhängig verteilt werden.

Praktisch wird dies so umgesetzt, dass es eine Liste von geordneten Paaren an Identifikationen für Vorlesungen gibt. Es werden all jene Vorlesungen gesucht, deren Identifikationen denen der Listenpaaren entsprechen. Für diese gilt nun, dass die Beginnzeit des

Nachfolgers größer als die des Vorgängers ist und außerdem auch nicht zu spät danach sein soll. Unmittelbar danach geht deshalb nicht, weil dies zu einem Widerspruch mit der zweiten Pause und der Modulobedingung an die Beginnzeit führen könnte. Außerdem müssen sie natürlich am selben Tag stattfinden.

Listing 3.37: unmittelbar nachfolgende Vorlesung

```
194 forall (n in nachfolger)
195  forall (ordered v1,v2 in rVOB: n.v1 = VOB[v1].ID & VOB[v2].ID = n.v2)
196   ((vo_zeit[v1]*15 + VOB[v1].dauer + pause[2] + (60 mod modul) >= vo_zeit[v2]*15)
197   & vo_zeit[v1] < vo_zeit[v2]
198   & vo_tag[v1]  = vo_tag[v2]);
```

Sollen die beiden Vorlesungen außerdem noch im selben Hörsaal stattfinden, so muss dies bei den Koppelungsbedingungen (siehe Seite 54) separat angegeben werden. Dies ist deshalb notwendig, damit wie im Beispiel eine Vorlesung mit einer anschließenden (Labor)-Übung in verschiedenen Räumen stattfinden kann.

Dies ist die entsprechende mathematische Formulierung:

$$
\begin{aligned}
\forall n \in \texttt{Nachfolger}: \quad & \forall v_1, v_2 \in \texttt{VO} : v_1 = n_{v_1} \wedge v_2 = n_{v_2} : \\
& v_1.\texttt{Zeit} \cdot 15 + v_1.\texttt{Dauer} + \texttt{Pause}_2 + (60 \mod \texttt{modul}) \geq v_2.\texttt{Zeit} \cdot 15 \\
\wedge \quad & v_1.\texttt{Zeit} < v_2.\texttt{Zeit} \\
\wedge \quad & v_1.\texttt{Tag} = v_2.\texttt{Tag}.
\end{aligned}
$$

3.3.3.4.10 Blocklehrveranstaltungen Die schon in Kapiel 2.2.2 beschriebenen Blocklehrveranstaltungen werden hier auf eine etwas komplexere Art umgesetzt. Blocklehrveranstaltungen zeichnen sich dadurch aus, nicht das ganze Semester lang durchgehenden gehalten zu werden, sondern nur für einige Wochen. Die Modellierung der Blöcke ist wie folgt zu verstehen: Zuerst betrachten wir das gesamte Semester. Da sich eine Lösung immer nur auf eine Woche bezieht, muss das Semester in Wochen eingeteilt werden um es als ganzes modellieren zu können. Dies kann durch eine einfache alphanumerische Kennzeichnung wie „W01“, „W02“, „W03“, ... geschehen. Bei genauerer Betrachtung wird aber schnell klar, dass sich zwei oder mehr Wochen mit genau identischen Vorlesungen zu einer Gruppe zusammenfassen lassen. Diese Gruppen wiederum sind genau die hier im Modell verwendeten Blöcke und sind deshalb eine Vereinfachung einer kompletten Auflistung aller Wochen. Es werde nur die relevanten Informationen verwendet. Natürlich lässt sich dies vernachlässigen und jede Woche einzeln behandeln. Dies macht zwar im Modell keinen Unterschied, wohl aber für die Anzahl der Randbedingungen und vor allem auch für die Menge an Daten die eingegeben werden müssen (pro Vorlesung die Anzahl boolescher Variablen die es Blöcke gibt). Eine Grafik in Abbildung 3.1 veranschaulicht diesen Umstand.

Die Umsetzung gestaltet sich so, dass allen Vorlesungen die Teile des Semesters, an denen sie gehalten werden, als Menge in den Daten angegeben wird. Gibt es also zum Bei-

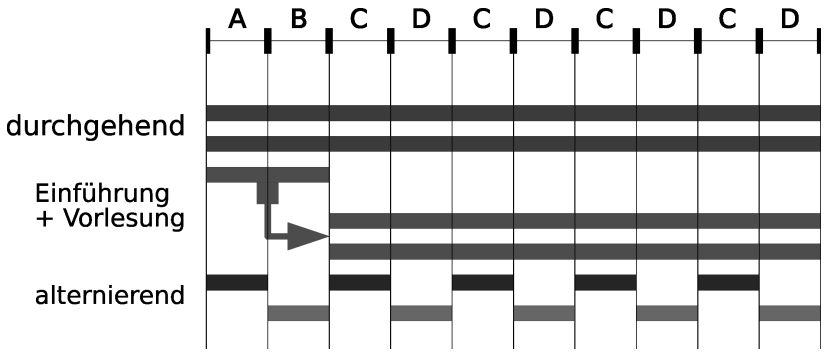


Abbildung 3.1: Unterteilung der Blocklehrveranstaltungen

spiel die Teile `A, B, C, D`, so handelt es sich um eine durchgehende Vorlesung wenn ihr die Menge { `A B C D` } zugeordnet ist. Die nur am Beginn stattfindenden Blocklehrveranstaltungen sind durch die Menge { `A B` } und die alternierenden Vorlesungen durch die Menge { `A C` } bzw. { `B D` } charakterisiert.

Folglich ist nur dann auf Überschneidungen der Vorlesungen zu achten, wenn sie sich auch im selben Block befinden. Die Einführungsvorlesung kann also durchaus im selben Hörsaal, am selben Tag und zur selben Zeit wie die zugehörige Semestervorlesung gehalten werden. Entsprechend gilt das auch für die sich abwechselnden Vorlesungen.

Aus den angegebenen Mengen in den Daten wird nun in einem Schritt vor der eigentlichen Formulierung des Problemes eine Matrix erzeugt, deren Zeilen und Spalten durch die Vorlesungen indiziert sind und die Einträge die Durchschnittsmengen der beiden zum Element gehörenden Vorlesungen sind. Der entsprechende Codeteil sieht so aus:

Listing 3.38: Überschneidungsmatrix der Blocklehrveranstaltungen initialisieren

```
199 {string} ueberschneidB[rVOB,rVOB];
200 initialize
201    forall(i,j in rVOB)
202       ueberschneidB[i,j] = VOB[i].BGruppe inter VOB[j].BGruppe;
```

In Zeile 199 wird die Matrix definiert und im darauf folgenden Initialisierungsdurchlauf werden in der Zeile 202 ihre Elemente eingetragen. Die mathematische Formulierung des Initialisierungsdurchlaufs sieht so aus:

$$\forall i, j \in [1, |\mathtt{VO}|]: \quad \mathtt{ueberschneidB}_{i,j} = \mathtt{VO}_i.\mathtt{Gruppe} \cap \mathtt{VO}_j.\mathtt{Gruppe}.$$

Nachdem das erledigt ist, sind all diejenigen Vorlesungen irgendwann im selben Abschnitt, wenn die Kardinalität der jeweiligen Menge in dieser Überschneidungsmatrix ≥ 1 ist. Das lässt sich nun unkompliziert testen (siehe Zeile 203) und erklärt die noch nicht beschriebenen Zeilen in den vorhergehenden Codefragmenten der anderen Randbedingungen.

Listing 3.39: Kardinalität der Überschneidungsmatrixeinträge

```
203 card(ueberschneidB[v1,v2]) >= 1
```

Als angenehmer Nebeneffekt erübrigt sich die Bedingung, dass die Vorlesungen in jedem ihrer Blöcke zur selben Zeit und im selben Raum stattfinden sollen, weil nur mit einer Variable über die Blöcke hinweg pro Vorlesung und gesuchtem Parameter gearbeitet wird.

3.3.3.4.11 Gemeinschaftlich gehaltene Vorlesungen Bei gemeinschaftlich gehaltenen Vorlesungen sind bei derselben Lehrveranstaltung mehrere Vortragende gleichzeitig anwesend. Dabei können mehrere Fälle auftreten: entweder müssen alle, nur mindestens einer oder eine bestimmte Anzahl der Vortragenden zugegen sein. Hier wird ähnlich wie in dem Abschnitt über einen zusätzlichen Raum nur der Fall genauer betrachtet, wenn alle Vortragenden anwesend sein müssen.

Die technische Umsetzung sieht so aus, dass es eine Liste von sogenannten *Meta-Vortragenden* gibt, die jeweils aus mehreren normalen Vortragenden bestehen. Es wird dann in der Beschreibung der Daten der Vorlesung dieser *Meta-Vortragende* als Vortragender angegeben, wobei seine Bezeichnung mit dem Präfix `meta` beginnt. Das dient zur Unterscheidung von den normalen Vortragenden.

Es müssen mehrere Bedingungen erfüllt werden, damit sich die Einteilung für alle anwesenden Vortragenden ausgehen kann.

1. Die jeweilige Vorlesung muss zu einer Zeit eingeteilt werden, zu der alle assoziierten Vortragenden Zeit haben.

2. Alle assoziierten Vortragenden können zur gewählten Zeit keine andere Vorlesung halten.

3. Keiner der zur jeweiligen Vorlesung assoziierten Vortragenden kann einer anderen auch von ihm gemeinschaftlich gehaltenen Vorlesung beiwohnen.

Der erste Punkt wird durch Listing 3.40 implementiert. Es handelt sich dabei im wesentlichen um eine Iteration über alle Vortragenden aller Vorlesungen. Die `und`-Bedingung ist durch das `forall`-Statement in Zeile 206 gegeben. Zeile 204 bis 206 sind dafür zuständig, die entsprechenden *Meta-Vortragenden* herauszufiltern, indem sie zuerst nur diejenigen Vorlesungen auswählen, deren Eintrag für den Vortragenden mit dem Prefix `meta` beginnt. Das Reduziert den Rechenaufwand. Dann wird aus der Liste aller *Meta-Vortragenden* dasjenige Element genommen, dessen `name` identisch mit dem Namen des Vortragenden der momentan iterierten Vorlesung ist. Nun wird nur noch in Zeile 206 über diese Vortragenden iteriert und in der darauffolgenden Zeile 207 der dazugehörige Eintrag in der Liste aller Vortragenden gesucht. Diese enthält dann die Information, zu welchen Zeiten der Vortragende verfügbar ist und schränkt so die Zeit für diese besonderen Lehrveranstaltung mit mehr als einem Vortragenden ein.

Listing 3.40: jeder Vortragende muss Zeit haben

```
204 forall(v in rVOB : VOB[v].profID.subString(0,4) = "meta")
205  forall(m in metaProf : VOB[v].profID = m.name)
206   forall(p in m.profID)
207    forall(ps in rProf : PR[ps].name = p)
208     forall(t in Tage)
209       (vo_tag[v]=t) =>
210        ( PR[ps].zeit[t].von <= vo_zeit[v]
211        & PR[ps].zeit[t].bis * 15 >=
212                          vo_zeit[v] * 15 + VOB[v].dauer * akaH);
```

Dies war erstmal die Bedingung an die zeitliche Position der jeweiligen Veranstaltung in Bezug auf die allgemeine Verfügbarkeit der dazugehörigen Vortragenden. Es muss ebenfalls eine Überschneidung zwischen den anderen Vorlesungen der einzelnen *Meta-Vortragenden* mit diesen *Meta-Vorlesungen* unterbunden werden.

Die entsprechende mathematische Formulierung ist hier etwas vereinfacht wiedergegeben:

$$\begin{aligned} \forall v \in \texttt{VO}: \quad & v \text{ wird von meta-Professoren } p \text{ gehalten} : \forall t \in \texttt{Tage}: \\ & v.\texttt{Tag} = t \Rightarrow p.\texttt{Zeit.von} \leq v.\texttt{Zeit} \\ \wedge \quad & p.\texttt{Zeit.bis} \cdot 15 \geq v.\texttt{Zeit} \cdot 15 + v.\texttt{Dauer} \cdot \texttt{akaH}. \end{aligned}$$

Folgendes Listing 3.41 behandelt die Verknüpfung zwischen normalen und *Meta-Vortragenden*. Zeile 213 bis Zeile 216 haben ähnlich wie im vorhergehenden Listing den Zweck, nur genau die entscheidenden Lehrveranstaltungen und Vortragenden herauszufiltern. Dies geschieht hier auf die Weise, dass zuerst in Zeile 213 die eine zu vergleichende Vorlesung aus denen der *Meta-Vorlesungen* gewählt wird, anschließend der zu ihr gehörige *Meta-Vortragende* dessen `name` mit dem Namen des Vortragenden der momentanen Vorlesung ident ist. Für alle Vortragenden dieses *Meta-Vortragenden* wird wieder iteriert und dann in Zeile 216 die zweite Vorlesung gewählt. Diese Vorlesungen sind nun all diejenigen, die als Vortragenden den momentanen Vortragenden aus der Aufzählung der Liste der *Meta-Vortragenden* aller *Meta-Vorlesungen* haben. Genau auf diese wird – nur wenn sie sich im selben Blockabschnitt befinden (siehe Seite 59, Absatz *Blocklehrveranstaltungen*)– die Überschneidungsbedingung angewendet.

Listing 3.41: keine Überschneidung zwischen allen Vorlesungen der Vortragenden

```
213 forall(v1 in rVOB : VOB[v1].profID.subString(0,4) = "meta")
214  forall(m in metaProf : VOB[v1].profID = m.name)
215   forall(p in m.profID)
216    forall(v2 in rVOB : VOB[v2].profID = p
217                      & card(ueberschneidB[v1,v2]) >= 1)
218       (vo_tag[v1]=vo_tag[v2]) =>
219   (vo_zeit[v1]*15 + VOB[v1].dauer*akaH + pause[1] <= vo_zeit[v2]*15
220 \/ vo_zeit[v2]*15 + VOB[v2].dauer*akaH + pause[1] <= vo_zeit[v1]*15);
```

Damit werden Punkt 1. und 2. durch die Lösung sicher erfüllt – es fehlt noch Punkt 3. Dieser braucht ähnlich wie der Absatz *Blocklehrveranstaltungen* auf Seite 59 zuerst eine Hilfsmatrix, deren Generierung im Listing 3.42 zu sehen ist. Sie besteht im Wesentlichen zuerst aus der Umwandlung der Menge der *Meta-Vortragenden* in einen indizierten Vektor, dessen Dimension die Größe der Matrix festlegt. (Nur die Menge der *Meta-Vortragenden* an sich kann nicht als Index dienen.) Anschließend (ab Zeile 226) werden die Einträge der Überschneidungsmatrix generiert.

Listing 3.42: Vorverarbeitung der Meta-Vortragenden

```
221 range rMP [1..card(metaProf)];
222 sMetaProf MP [rMP];
223 initialize forall(i in rMP)
224    MP[i] = metaProf.next(metaProf.first(),i-1);
225 {string} MPueberschneid[rMP, rMP];
226 initialize
227    forall(i,j in rMP)
228       MPueberschneid[i,j] = MP[i].profID inter MP[i].profID;
```

Gerüstet mit der Überschneidungsmatrix folgt nun die Implementierung der 3. Bedingung. Listing 3.43 zeigt die doppelt verschachtelte Schleife mit der bereits bekannten Überschneidungsbedingung im Kern. Zuerst wird in der äußeren Schleife ein geordnetes Paar von jeweils unterschiedlichen Vorlesungen gewählt, die beide von einem virtuellen *Meta-Vortragenden* gehalten werden. Die innere Schleife wählt nun aus allen *Meta-Vortragenden* ein Paar in der Art aus, dass einer des Paares der *Meta-Vortragende* der ersten gewählten Vorlesung und der andere Vortragende der anderen Vorlesung ist. Damit erhält man genau die Referenzen auf die beiden *Meta-Vortragenden* der betrachteten Vorlesungen. Von denen sind diejenigen von Interesse, die einen gemeinsamen Vortragenden haben – genau so einer kann zur selben Zeit am selben Tag nur einer Lehrveranstaltung beiwohnen (Zeile 234), außer natürlich, es handelt sich um zwei Vorlesungen die nie im selben Block gehalten werden! Dies filtert die aus dem Abschnitt über *Blocklehrveranstaltungen* (siehe Seite 59) bekannte Bedingung in Zeile 231 heraus!

Listing 3.43: keine Überschneidungen bei unterschiedlichen Meta-Vortragenden

```
229 forall(ordered v1,v2 in rVOB : VOB[v1].profID.subString(0,4) = "meta"
230                              & VOB[v2].profID.subString(0,4) = "meta"
231                              & card(ueberschneidB[v1,v2])    >= 1 )
232   forall(mp1,mp2 in rMP : MP[mp1].name=VOB[v1].profID
233                         & MP[mp2].name=VOB[v2].profID
234                         & card(MPueberschneid[mp1,mp2]) >= 1)
235      (vo_tag[v1]=vo_tag[v2]) =>
236  ( vo_zeit[v1]*15 + VOB[v1].dauer*akaH + pause[3] <= vo_zeit[v2]*15
237 \/ vo_zeit[v2]*15 + VOB[v2].dauer*akaH + pause[3] <= vo_zeit[v1]*15);
```

Die mathematische Formulierung ist weitgehend identisch mit all denjenigen, die ebenfalls Überschneidungen zwischen Vorlesungen ausschließen. Einzig die im vorhergehen-

den Listing 3.42 eingeführte Matrix `MPueberschneid` ist neu. Diese hat denselben Zweck wie die Überschneidungsmatrix bei den Blocklehrveranstaltungen und wird nach Listing 3.38 erklärt.

3.3.3.4.12 Verfügbarkeit der Vortragenden Von der Struktur verhält es sich mit den Vortragenden recht ähnlich wie mit den Hörsälen. Im Unterschied dazu sind sie aber den Vorlesungen eindeutig zugeordnet. Dadurch entfällt die indirekte Indizierung. Ein Vortragender kann nur eine Vorlesung zur selben Zeit halten und natürlich nur dann, wenn er dafür Zeit hat.

Das folgende Listing 3.44 schränkt die Wahl der Vorlesungszeiten auf die am jeweiligen Tag möglichen für jeden der Vortragenden ein. Indem über alle Vorlesungen, die von keinem *Meta-Vortragenden* gehalten werden, iteriert wird, erhält man alle Referenzen auf die normalen Vortragenden. In deren Daten steht die Zeit, an der sie gewählt werden darf.

Listing 3.44: Randbedingung der Verfügbarkeit der Vortragenden

```
238 forall(i in rVOB)
239   forall(p in rProf : PR[p].name = VOB[i].profID
240                     & VOB[i].profID.subString(0,4) <> "meta")
241     forall(t in Tage)
242       (vo_tag[i]=t) =>
243 ( PR[p].zeit[t].von <= vo_zeit[i]
244 & PR[p].zeit[t].bis * 15 >= vo_zeit[i] * 15 + VOB[i].dauer * akaH);
```

Außerdem gibt es noch weitere Beschränkungen: ausgeschlossene Zeitintervalle und eine Mittagspause.

Wie bei den Hörsälen gibt es die Möglichkeit, bestimmte Zeitintervalle auszuschließen. Hier kann man sich genauso vorstellen, dass ein Vortragender für eine gewisse Zeit eine andere nicht durch dieses Modell erfasste Vorlesung hält oder seine Sprechstunde hat.

Dies wird hier auf die Art realisiert, dass zuerst über alle normalen Vorlesungen iteriert wird und dann mit der Referenz auf den Vortragenden bestimmt wird, welche Zeiten auszuschließen sind. Gibt es für ein und denselben Vortragenden mehrere Ausschlusszeiten, stellt das kein Problem dar und wird daher korrekt behandelt.

Listing 3.45: Ausschlusszeiten für Vortragende

```
245 forall (v in rVOB : VOB[v].profID.subString(0,4) <> "meta")
246   forall (bp in profBelegt : bp.name = VOB[v].profID)
247     (vo_tag[v]=bp.tag) =>
248  (    (vo_zeit[v]*15 + VOB[v].dauer*akaH <= bp.von*15)
249   \/ (vo_zeit[v]                         >= bp.bis));
```

Die entsprechende mathematische Formulierung sieht so aus:

$\forall v \in \texttt{VO} : v$ wird von metaProfessoren p gehalten :

$\forall$ Beginnzeit b jedes metaProfessors p : $v.\texttt{Zeit} \cdot 15 + v.\texttt{Dauer} \cdot \texttt{akaH} \leq b.\texttt{von} \cdot 15$

$\vee$ $v.\texttt{Zeit} \geq b.\texttt{bis}.$

Allerdings muss das dann auch für die *Meta-Vortragenden* gelten, die im vorhergehenden Abschnitt 3.3.3.4.11 eingeführt wurden. Sie bestehen aus einzelnen Referenzen, über die iteriert werden muss. Sonst verhält es sich identisch wie zum vorhergehenden Listing 3.45.

Listing 3.46: Ausschlusszeiten für *Meta-Vortragende*

```
250 forall (v in rVOB : VOB[v].profID.subString(0,4) = "meta")
251  forall (mp in metaProf : mp.name = VOB[v].profID)
252    forall (p in mp.profID)
253      forall (bp in profBelegt : bp.name = p)
254        (vo_tag[v]=bp.tag) =>
255 (   (vo_zeit[v]*15 + VOB[v].dauer*akaH <= bp.von*15)
256 \/ (vo_zeit[v]                         >= bp.bis));
```

Zusätzlich gibt es noch die Möglichkeit, eine *Mittagspause* anzugeben. Dabei wird in den Daten eine bestimmte Zeitspanne eingegeben, die der Mittagspause entspricht und zu jedem Vortragenden wird in einer booleschen Variable gespeichert, ob er sie machen möchte. Wenn dem so ist, darf in dieser Zeit ebenfalls keine ihm zugewiesene Vorlesung stattfinden.

Listing 3.47: optionale Mittagspause der Vortragenden

```
257 forall(i in rVOB : VOB[i].profID.subString(0,4) <> "meta")
258    forall(p in rProf : PR[p].name = VOB[i].profID
259                      & PR[p].mittagspause=1)
260       (vo_zeit[i]                        >= MP_zeit.bis)
261    \/ (vo_zeit[i]*15+VOB[i].dauer*akaH <= MP_zeit.von*15);
```

Selbiges für *Meta-Vortragende*:

Listing 3.48: optionale Mittagspause der Vortragenden

```
262 forall(i in rVOB : VOB[i].profID.subString(0,4) = "meta")
263  forall(mp in metaProf : mp.name = VOB[i].profID)
264   forall(ps in mp.profID)
265    forall(p in rProf : PR[p].name = ps
266                      & PR[p].mittagspause=1)
267       (vo_zeit[i]                        >= MP_zeit.bis)
268    \/ (vo_zeit[i]*15+VOB[i].dauer*akaH <= MP_zeit.von*15);
```

Die letzten beiden Paare von Bedingungen hätten sich mit einer `if-then-else`-Konstruktion vereinfachen lassen. Das würde keinen effektiven Unterschied machen, aber die Lesbarkeit vermindern.

Damit sind alle Bedingungen in der Modellierungssprache des ILOG-Solvers formuliert. Im nächsten Schritt wird eine Strategie formuliert, eine Lösung berechnet und analysiert.

3.3.4 Strategie

Anfangs habe ich mit einer kleinen Anzahl von Vorlesungen gearbeitet, um das Modell schneller aufbauen zu können, weil so jeder Lösungsschritt sehr schnell erledigt war. Auch lassen sich auf diese Art schnell viele pathologische Fälle durchtesten, um Fehler finden zu können. Viele der hier vorgestellten Bedingungen lassen sich auf viele Arten implementieren, wie auch die Struktur der Daten viel Spielraum offen lässt. Auf jeden Fall aber wächst der Aufwand der Lösungssuche mit steigender Anzahl von Vorlesungen. Ohne die Reduzierung auf nur eine einzige Zeitvariable im Viertel- Halbe- oder Ganze-Stunden Intervall ließen sich schon relativ kleine Probleme nicht mehr in vernünftiger Zeit (oder gar nicht, da der Speicherverbrauch plötzlich massiv anstieg und dadurch kein sinnvolles Rechnen mehr möglich war) lösen. Ziel war es jedoch, Lösungen für eine Vorlesungsplanung realer Größe zu erstellen.

Um hier bei größeren Datenmengen noch eine Lösung zu bekommen, lässt sich zusätzlich zum bisherigen Modell eine Suchstrategie angeben. In ihr fließen heuristische Überlegungen ein – siehe hierzu die Erläuterungen zu Variablenreihungen auf Seite 23. Konkret heißt das, es werden Strategien eingebaut, die bei der Suche nach der optimalen Lösung helfen sollen, oder Einschränkungen an den abzusuchenden Lösungsraum angegeben. Leider ist die Dokumentation nicht umfangreich genug, um alle meine Vorstellungen umsetzen zu können. Es reichte jedoch um das Problem überhaupt lösen zu können. Ohne der hier vorgestellten Strategie ist der ILOG-Solver bei den Tests niemals in der Lage gewesen, eine Lösung für größere Datensätze zu finden.

Die Idee der Strategie besteht aus zwei Teilen. Erstens werden die Vorlesungen in einer bestimmten Reihenfolge eingeteilt, wobei gleichzeitig jeder einzelnen Vorlesungsvariablen Zeit, Tag und Hörsaal (in genau dieser Reihenfolge!) zugewiesen werden. Zweitens ist diese Reihenfolge relativ streng festgelegt und ist der Praxis der Zeiteinteilung angelehnt. Es werden diejenigen Vorlesungen an den Anfang des Entscheidungsprozesses gereiht,

- die aus vielen Teilen bestehen,
- deren Teile synchron gesetzt werden (siehe *Teile zur selben Zeit – TZSZ* Seite 58),
- deren Teile auch am selben Tag sein dürfen (siehe *Teile auch am selben Tag – TAAST* Seite 57)
- die eine längere Dauer haben,
- die eine höhere Anzahl Zuhörer haben – weil mehr Zuhörer die Wahlfreiheit für den Hörsaal einschränken,

und abschließend nach der absteigenden Größe der Domains der Variablen.

Siehe hierzu auch den Abschnitt über Variablenzuweisung auf Seite 23.

Diese Ordnungsrelation und Zuweisungsreihenfolge wird in folgendem Codeteil erklärt:

Listing 3.49: eine mögliche Strategie

```
forall(v in rVOB ordered by decreasing <
               VOBanzTeile[v],
               TZSZ.member(VOB[v].ID.name),
               TAAST.member(VOB[v].ID.name),
               VOB[v].dauer,
               VOB[v].MaxAnzZH,
               -dsize(vo_zeit[v])-dsize(vo_tag[v])-dsize(vo_hs[v])>) {
   if not bound(vo_zeit[v]) then generate(vo_zeit[v]) endif;
   if not bound(vo_tag[v])  then generate(vo_tag[v])  endif;
   if not bound(vo_hs[v])   then generate(vo_hs[v])   endif;
  };
};
```

Außerdem wird als Suchalgorithmus die „Depth-First Strategy“ gewählt, weil sie am Anfang die Variablen für die Zielfunktion auf `0` setzt und so den Lösungsraum größer lässt. Dies dauert länger, führt aber eher zu einer Lösung, die dann im weiteren Suchverlauf optimiert wird. Alternativ funktioniert auch die „Best-First-Strategy“, welche aber oft länger braucht, um eine erste Lösung zu finden. Dafür sollte sie früher einen besseren Wert für die Zielfunktion liefern. Alle anderen Suchstrategien haben bei den Tests eher schlecht abgeschnitten. Zumindest auf die Art wie sie konfiguriert wurden suchten sie zu viel in der „Breite“, d. h. sie kamen nie so weit, allen Variablen Werte zuzuweisen, sondern testeten teilweise besetzte Konfigurationen intensiv durch. Abbildung 3.2 zeigt ein Beispiel eines Suchbaums für die „Best-First-Strategy“. (Abbildung 3.2) ist eine Detailansicht mit der Lösung an der Spitze)

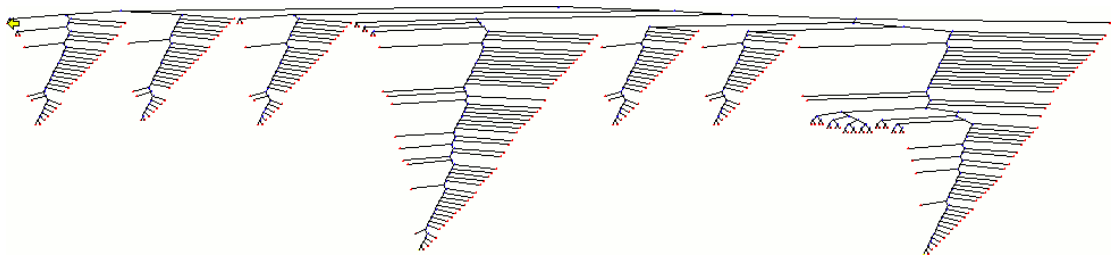


Abbildung 3.2: Suchbaum „Best-First-Strategy“ – Totalansicht

In der Praxis zeigte sich jedoch, dass es eine mir nicht nähere bekannte Limitierung gibt, welche häufig bei großen Datenmengen einer vollständigen Lösung im Weg steht. Nach einiger Zeit wird der Fehler `no more memory segments` ausgegeben. An dieser Stelle bleibt nur zu hoffen, dass vor diesem Fehler schon eine noch nicht optimale Lösung gefunden wurde, die dann weiter verwendet werden kann. Bei meinen Testdaten für ein

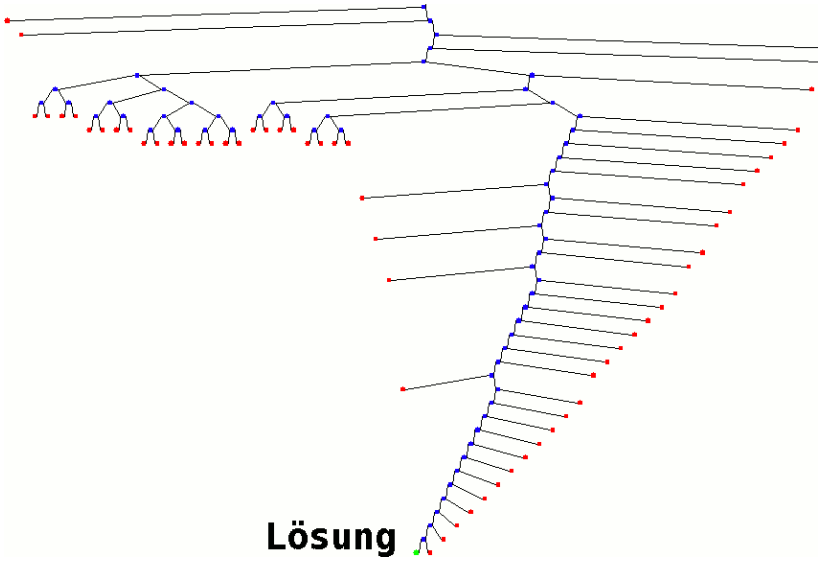


Abbildung 3.3: Suchbaum „Best-First-Strategy“ – Detail

realistisches Problem konnte fast immer eine Lösung gefunden werden, und nach etwa 20 Iterationssschritten kam es zur besagten Fehlermeldung. Führen die Daten rasch auf einen Widerspruch, so bricht der Lösungsvorgang sehr schnell ohne Lösung ab.

3.3.5 Behandlung von Widersprüchen - „Soft-Constraints“

Im Zuge der Entwicklung und den parallel laufenden Tests des Modells stellte sich bald heraus, dass es Situationen gibt, in denen es aufgrund von Widersprüchen der Bedingungen keine Lösung gibt. Sei es nur, dass eine Vorlesung verpflichtend zu einer Zeit gesetzt wird, an der der vortragende Professor keine Zeit hat, oder etwas komplexer, wenn die Kombination mehrerer Bedingungen untereinander nicht erfüllbar sind. Ein Beispiel für Letzteres wäre etwa die Bedingung, dass die Teile einer Vorlesung zu unterschiedlichen Tagen sein müssen, wenn dies für ein in sechs Teilen gehaltenes Proseminar gilt, wobei die Woche aber nur fünf Tage hat. Noch komplexere Kombinationen sind natürlich auch möglich und müssen bei jeder Lösung erfüllt sein. Dies erledigt der Solver automatisch durch Fortpflanzen der Nebenbedingungen.

Hierbei entsteht jedoch folgendes Problem: Kann keine Lösung gefunden werden, so gibt der ILOG Solver als Informationsmeldung nur aus, dass es keine Lösung gibt. Es ist daher nicht möglich, direkt herauszufinden, warum keine Lösung gefunden wurde. Klarerweise wird es nicht gelingen, bei einer komplexen Kombination aus vielen Bedingungen anzugeben, warum es nicht funktioniert. Es kann jedoch schon eine Auflistung der nicht erfüllbaren Bedingungen weiterhelfen zu verstehen, wo das Problem liegen könnte. Vor allem bei kleinen, einfachen Bedingungen wäre es sehr praktisch eine Rückmeldung zu haben, um unabsichtliche Fehler in der Dateneingabe zu vermeiden. Hierbei gilt es aber zu bedenken, dass implizit von unterschiedlich wichtigen Bedingungen gesprochen wird. Gemeint ist, dass zum Beispiel eine erzwungene Beginnzeit einer Vorlesung, die

im Widerspruch zur möglichen Zeit des Vortragenden steht, als stärker einschränkend empfunden wird – der Solver jedoch bewertet beide Bedingungen gleich. Ist also nun die Einschränkung auf die genaue Beginnzeit oder die auf die verfügbare Zeit des Vortragenden diejenige Bedingung, die nicht erfüllt werden konnte? Im Allgemeinen kann das nicht geklärt werden, da beide einen irrtümlichen Fehler beinhalten können. Es kann jedoch so vorgegangen werden, dass in einem gewissen Sinn über die Gesamtzahl an gefundenen Diskrepanzen minimiert wird. Dadurch entsteht eine minimale Menge von möglichen Fehlern oder Widersprüchen des Modells – diese muss aber nicht zwingend das eigentliche Problem enthalten.

Nach einigen Überlegungen wurde klar, dass es mehrere Ansätze geben könnte, um trotzdem zu einer nützlichen Information über die ungefähre Ursache einer misslungenen Lösungssuche zu kommen. Die erste Idee war, die Einschränkungen an den Lösungsraum der einzelnen Variablen zu untersuchen. Hierfür, so war meine Idee, müsste es möglich sein, den während der Lösungssuche noch freien Wertebereich herauszufinden. Die Variablen, bei denen beim Abbrechen der Lösungssuche der noch zur Verfügung stehende Bereich leer ist, konnten nicht mit passenden Werten belegt werden. Deshalb wären diese Variablen Teil des Problems. Jedoch fand ich keine Möglichkeit, diese Information über das API herauszufinden. Laut Webseite des ILOG Solvers gibt es in einer späteren Version des Solvers die Möglichkeit, sogenannte Soft-Constraints zu verwenden. Das sind Bedingungen, die nicht unbedingt erfüllt sein müssen. Über sie lässt sich auch herausfinden, warum es dem Solver nicht gelang das Problem zu lösen. Diese spätere Version steht mir nicht zur Verfügung und daher erarbeitete ich eine Methode, diese Technik in das Modell einzubauen. Ausgangspunkt für diese Überlegungen war, jede Gruppe von Bedingungen durch `oder`-Verknüpfungen mit einer booleschen Variable zu versehen. Können alle diese zusätzlichen Variablen auf `0` gesetzt werden, so sind alle erfüllbar, und das Modell hat eine Lösung. Können nicht alle erfüllt werden, so geben die Indizes im Vektor dieser Hilfsvariable an, welche der Randbedingungen nicht erfüllt werden konnten. Dies ist aber leider nur eine sehr grobe Information, da nicht klar ist, welche der vielen Vorlesungen das Problem verursacht hat.

Ein weiterer Ansatz war, die Menge der Vorlesungen zu verkleinern. Es wird also versucht, ein viel kleineres Problem zu lösen. Ist dies möglich, so muss das Problem bei den nicht betrachteten Vorlesungen begraben liegen. Auf diese Art ließe sich ein Suchalgorithmus konstruieren, der eine minimale Menge an Vorlesungen ausgibt, die zur Unlösbarkeit des Problems führten. Weil aber jede der Vorlesungen mit anderen Bedingungen verknüpft ist, müsste für jede Teilmenge der Vorlesungen immer auf die exakte Konsistenz der Daten geachtet werden. Das wäre ein relativ hoher Aufwand für eine vermutlich nicht sehr zufriedenstellende Methode, weil sie sehr lange suchen würde und trotzdem nicht besonders informativ ist. Es würde nämlich bei einer Teilung einer Menge mit mehreren problematischen Vorlesungen ein Teil wegfallen können, der solche Vorlesungen enthält. Im Endeffekt bekommt man also nur eine Teilmenge der gewünschten Vorlesungen und müsste den Prozess öfters wiederholen.

Eine Kombination aus beiden Ideen würde darin bestehen, in jeder Gruppe von Bedingungen für jede Vorlesung eine Boolesche Variable anzugeben. Das würde dann ein

Eintrag aus einer sehr großen Matrix sein, die über die Menge der Constraints und die Menge aller Vorlesungen indiziert ist. Anfangs schien mir dies nicht machbar, weil es eine so große Menge an Variablen ist. Tests zeigten jedoch, dass dies mit starken Einschränkungen an diese Matrix doch machbar ist. Der Grund, warum es funktioniert, ist der, dass jede dieser Hilfswerte in dieser Matrix nur an einer einzigen Stelle im Modell verwendet wird, mit nur einer oder ein paar Variablen verknüpft ist und an diese Matrix nur ganz wenige, aber sehr starke Bedingungen gestellt werden. Deshalb dienen diese booleschen Werte als Auffanglager für alle Randbedingungen, die nicht erfüllt werden können aber durch die `oder`-Verknüpfung im ganzen erfüllt sind. Der Vorteil dieser Matrix ist dann der, dass alle Einträge mit `1` genau angeben, welche Bedingungen bei welchen Vorlesung Probleme machen.

Bei der Entwicklung der genauen Implementation dieser Idee ergaben sich aber sehr schnell einige Probleme. Mein erster Ansatz bestand darin, erstmal nur eine Bedingung mit einem Vektor für die Vorlesungen zu versehen. In Tests wurden dann dazu passende fehlerhafte Daten angegeben und getestet, ob es funktioniert. Leider hatte das nur mäßigen Erfolg, da nicht nur die eine Vorlesung, sondern sehr viele zurückgegeben wurden. Also musste eine Einschränkung an die maximale Anzahl von `1`-Werten in dem Vektor angegeben werden. Das entsprach im Modell einer Kleinergleich-Bedingung über die Summe aller Einträge des Vektors. Dies jedoch ist natürlich nicht praktikabel, wenn mehr Probleme als maximal aufgefangen werden können, vorhanden sind. Außerdem ist die Zeit, bis eine Lösung gefunden wird, sehr groß. Tests haben gezeigt, dass wenn nur eine einzige fehlerhafte Angabe in den Daten ist, diese nicht gefunden wird. Der Solver bricht bei dieser und ähnlichen Situationen häufig mit der schon erwähnten Meldung `no more memory segments` ab ohne eine Lösung gefunden zu haben.

Eine dennoch funktionierende Weiterentwicklung war, diese über Vorlesungen und Constraints indizierte Matrix weiteren Einschränkungen zu unterwerfen. Dabei beschränkte ich die Summe über alle Vorlesungen eines Constraints durch einen fixen Wert (`2` bis `4`) und führte einen Variablenvektor ein, der in jedem Eintrag eine Ganzzahl mit Wertebereich von `0` bis zu diesem fixen Wert hat. Außerdem deaktivierte ich die Suche nach optimaler Wunscherfüllung in der Zielfunktion, indem ich sie mit `0` multiplizierte und minimierte alle Werte dieses Variablenvektors. Dies war relativ erfolgreich, erklärte ich doch ziemlich genau, dass ich bei einer stark eingeschränkten Matrix möglichst wenig Probleme finden möchte. Bei einem Maximalwert von `2` bedeutete dies, dass er bei ca. 60 Widersprüchen anfing und dann etwa auf 50 minimierte. Danach kam die bereits bekannte Meldung `no more memory segments` und er brach ab. Die zuletzt gefundene Lösung war aber noch verwertbar. Eine weitere erzwungene Einschränkung an die Summe über alle Werte in diesem Variablenvektor zwang nun den Solver, gleich bei den zum Beispiel angegeben Maximalwert von `20` Diskrepanzen mit der Minimierung zu beginnen – sofern er aber überhaupt eine Lösung fand. Werden bei nur einer Problemstelle sehr weite Suchparameter angegeben, findet er viel zu viele oder sucht zu lange; gibt es viele Problemstellen, so findet er bei zu engen Suchparametern gar keine Lösung.

Folgendes Listing 3.50 zeigt den Code für diesen Ansatz. Die ersten Zeilen sind ein Beispiel für die Erweiterung einer der 30 Bedingungen, die mit solchen Softconstraints

ausgestattet wurden. Dann die Variablen und die eben erwähnten Bedingungen an die Variablen und die Bedingung selber.

Listing 3.50: Softconstraints 1

```
281 forall(i in rVOB)
282   (VOB[i].MaxAnzZH <= HSMaxAnzZuhoerer[vo_hs[i]]) \/ softC[15,i];
283
284 var Boolean softC[rSoftC, rVOB];
285 range rSoftC_Thld [0..SoftC_Thld_max];
286 var rSoftC_Thld SoftC_Thld[rSoftC];
287
288 sum (j in rSoftC)   SoftC_Thld[j] <= SoftC_Thld_sum;
289       forall(j in rSoftC)
290          sum(i in rVOB) softC[j,i] <= SoftC_Thld[j];
```

Hierbei ist in `SoftC_Thld_max` der Maximalwert für den Wertebereich der Variablen des Vektors angegeben und `SoftC_Thld_sum` der Maximalwert für die Summe über die Werte dieses Vektors. `softC` enthält die Matrix, ihr erster Index ist für die Constraints und der zweite für die Vorlesungen. Es lässt sich in fast allen Bedingungen nach diesem Schema diese Matrix verwenden, auch solche, die zwei Vorlesungen koppeln (siehe Kapitel 3.3.3.4.9 auf Seite 54). Hier wird die erste Vorlesung als Index genommen.

Nach einigen Tests zeigte sich, dass immer auch Widersprüche ausgegeben werden, die kein Problem sind. Daher überlegte ich, wie ich noch weitere Einschränkungen machen könnte, die trotzdem noch funktionieren und sich genau auf das gesuchte konzentrieren. Eine Variante bestand darin, einen zusätzlichen Wert anzugeben, welche der Randbedingungen relaxiert werden sollen, wobei gleichzeitig alle anderen als normal (also alle erfüllt) behandelt werden. Nur eine Randbedingung zu relaxieren und alle anderen zu vernachlässigen ist auch möglich. Dies funktioniert jedoch nicht, da jede Bedingung erst auch durch die anderen Sinn macht. Es ist dann aber so, dass nur wenn alle anderen Randbedingung vollständig erfüllbar sind, eine Lösung gefunden werden kann. Daher ist diese Methode nicht geeignet, Probleme in zwei disjunkten Constraintgruppen herauszufinden.

Eine entsprechende Erweiterung des oben aufgeführten Codes sieht so aus, dass eine Variable `SoftC_whichone` angegeben werden muss. Ist ihr Wert `0` so wird wie oben verfahren, ist ihr Wert größer als `0`, so wird genau die Randbedingung mit diesem Wert relaxiert, alle anderen vollständig erzwungen.

Listing 3.51: Softconstraints 2

```
291 if(SoftC_whichone = 0) then {
292       sum (j in rSoftC)   SoftC_Thld[j] <= SoftC_Thld_sum;
293       forall(j in rSoftC)
294          sum(i in rVOB) softC[j,i] <= SoftC_Thld[j];
295 } else {
296   forall(j in rSoftC, i in rVOB: not(j=SoftC_whichone))
```

```
297       softC[j,i] = 0;
298     forall(j in rSoftC)
299       sum(i in rVOB) softC[j,i] <= SoftC_Thld[j];
300 }
301 endif;
```

Im Fall `SoftC_whichone` gibt es nur eine Möglichkeit das Modell zu erfüllen. Das durch ein Programm angedachte ausprobieren des richtigen Wertes kann auch der Solver übernehmen. Es muss also eine weitere Variable eingeführt werden, die angibt, in welcher der Gruppen von Randbedingungen nach dem Widerspruch gesucht werden soll. Der boolesche Wert in `SoftC_allbutone` gibt an, ob wie in Listing 3.50 oder auf die beschriebene Art gesucht werden soll.

Listing 3.52: Softconstraints 3

```
302 if(SoftC_allbutone = 0) then {
303  sum (j in rSoftC)   SoftC_Thld[j] <= SoftC_Thld_sum;
304  forall(j in rSoftC)
305    sum(i in rVOB) softC[j,i] <= SoftC_Thld[j];
306  SoftC_whichone = 0;
307 } else {
308  forall(j in rSoftC)
309    sum(i in rVOB) softC[j,i] <= SoftC_Thld[j];
310  forall(j in rSoftC, i in rVOB)
311    (not(j=SoftC_whichone)) => SoftC_Thld[j] = 0;
312 }
313 endif;
```

Ideal wäre nun eine Methode, dies auch für mehrere Gruppen von Randbedingungen zu machen. Diejenige von Listing 3.50 ist ein sehr guter Ansatz, jedoch etwas zu allgemein. Listing 3.52 funktioniert aber viel besser, um Fehler in einer einzigen Randbedingung zu finden. Es muss allerdings angemerkt werden, dass die Obergrenze `SoftC_Thld_max` für Variable `SoftC_Thld` weit höher angesetzt werden muss. Tests zeigten, dass ein Wert um die 5 recht schnell einen einzigen Fehler findet. Der Wert der Zielfunktion, der der Anzahl von vermuteten Problemen entspricht, wird im Rahmen der Optimierung von 5 auf etwa 3 minimiert. Es bleiben also nur sehr wenige Vorlesungen an genau bekannten Stellen für die manuelle Kontrolle übrig. Dies ist eine Verbesserung um mehr als den Faktor 10 zum ersten Ansatz.

Gilt es mehrere Probleme zu finden, so bleibt es weiterhin weitaus schwieriger, ebenso gute Resultate zu finden.

3.4 Lösung

In diesem Abschnitt stelle ich eine mögliche Lösung der bis hierher beschriebenen Daten und Randbedingungen vor. Das ist ein Vektor von Objekten, der innerhalb des OPL-

Modells definiert wurde. Die jeweiligen Einträge bestehen aus den kommaseparierten Werten, die jeder Vorlesung zugeordnet werden. Da mehrteilige Vorlesungen in separaten Teilen als jeweils eigenständige Vorlesung behandelt wurden, gibt es wie gewünscht mehrere Einträge pro Vorlesung mit aufsteigender Teilnummerierung. Außerdem ist nochmals der zugeordnete Vortragende erwähnt und abschließend die Werte der Variablen für den Tag, die Zeit und den gewählten Hörsaal. Nun kann – aufbauend auf diese Rohdaten – eine besser zu handhabende Ausgabe erstellt werden.

Listing 3.53: Beispiel einer Lösung des Modells

```
res = {
   <name:"anf1  ",teil:1,profID:"prof1",tag:Mon,zeit:32,hsaal:1>,
   <name:"anf1  ",teil:2,profID:"prof1",tag:Die,zeit:32,hsaal:1>,
   <name:"anf2  ",teil:1,profID:"prof2",tag:Mon,zeit:40,hsaal:1>,
   <name:"anf2  ",teil:2,profID:"prof2",tag:Die,zeit:40,hsaal:1>,
   <name:"einfA ",teil:1,profID:"prof1",tag:Mon,zeit:32,hsaal:1>,
   <name:"einfA ",teil:2,profID:"prof1",tag:Die,zeit:32,hsaal:1>,
   <name:"einfA ",teil:3,profID:"prof1",tag:Mit,zeit:32,hsaal:1>,
   <name:"pruefA",teil:1,profID:"unabh",tag:Fre,zeit:56,hsaal:4>,
   <name:"anfA1 ",teil:1,profID:"prof4",tag:Don,zeit:40,hsaal:1>,
   <name:"anfA1 ",teil:2,profID:"prof4",tag:Fre,zeit:40,hsaal:3>,
   <name:"anfA2 ",teil:1,profID:"prof1",tag:Die,zeit:52,hsaal:5>,
   <name:"anfA3 ",teil:1,profID:"prof4",tag:Mon,zeit:48,hsaal:1>,
   <name:"fortA1",teil:1,profID:"prof3",tag:Mon,zeit:40,hsaal:2>,
   <name:"fortA1",teil:2,profID:"prof3",tag:Don,zeit:40,hsaal:2>,
   <name:"fortA2",teil:1,profID:"prof4",tag:Mon,zeit:56,hsaal:2>,
   <name:"fort1 ",teil:1,profID:"prof3",tag:Mit,zeit:44,hsaal:5>,
   <name:"fort2 ",teil:1,profID:"prof2",tag:Mon,zeit:32,hsaal:2>,
   <name:"fort2 ",teil:2,profID:"prof2",tag:Die,zeit:32,hsaal:2>,
   <name:"fort3 ",teil:1,profID:"prof3",tag:Die,zeit:60,hsaal:5>,
   <name:"allgVO",teil:1,profID:"prof4",tag:Mit,zeit:48,hsaal:4>,
   <name:"allgVO",teil:2,profID:"prof4",tag:Don,zeit:48,hsaal:4>,
   <name:"PSFort",teil:1,profID:"prof2",tag:Don,zeit:36,hsaal:2>,
   <name:"PSAnf1",teil:1,profID:"prof1",tag:Mit,zeit:56,hsaal:4>,
   <name:"PSAnf2",teil:1,profID:"prof2",tag:Mit,zeit:56,hsaal:5>,
   <name:"PSAnf3",teil:1,profID:"prof3",tag:Mit,zeit:32,hsaal:1>,
   <name:"PSAnf4",teil:1,profID:"prof2",tag:Die,zeit:52,hsaal:3>,
   <name:"PSAnf5",teil:1,profID:"prof4",tag:Mon,zeit:64,hsaal:2>,
   <name:"VA1   ",teil:1,profID:"prof4",tag:Mon,zeit:72,hsaal:1>,
   <name:"VA1   ",teil:2,profID:"prof4",tag:Don,zeit:72,hsaal:1>,
   <name:"VA2   ",teil:1,profID:"prof3",tag:Die,zeit:68,hsaal:5>,
   <name:"VA2   ",teil:2,profID:"prof3",tag:Mit,zeit:68,hsaal:5>
}
```

3.4.1 Tests

Um das Modell und den Solver auf Praxistauglichkeit zu überprüfen, erweiterte ich den Datensatz Schritt für Schritt bis zu einem gesamten Semester. Dabei handelte es sich um die Vorlesungen und Proseminare an der Fakultät für Mathematik der Universität Wien, sowohl für Diplom als auch Lehramtsstudenten. Außerdem werden einige mathematische Vorlesungen für Physik ebenfalls von Vortragenden des Instituts für Mathematik gehalten, welche ich auch erfasste. Da der Ort dieser Vorlesungen weiter entfernt liegt, erwies sich die Randbedingung für Wegzeiten von Seite 45 als nützlich.

Die verwendeten Daten für diesen Test unter Realbedingungen sind dem Wintersemester 2005/06 ähnlich und im Anhang in Listing 7.4 auf Seite 128 nachzulesen. Ich habe großteils Standardwerte genommen und nur einige zusätzliche Bedingungen. Weiters gibt es einige völlig frei gewählte Wünsche. Die innerhalb von etwa 10 Sekunden, nach einigen sich inkrementell besserenden Iterationssschritten, ermittelte Lösung ist im Anhang in Listing 7.5 auf Seite 140 zu sehen. Auch ist ersichtlich, dass immerhin 13 Wünsche erfüllt werden konnten!

Daher ist es meiner Beobachtung nach möglich, nach wenigen Sekunden ein Ergebnis zu erhalten. Das ist relativ erstaunlich, da es sich der internen Statistik des Solvers nach um tausende Variablen und zehntausende Randbedingungen handelt. Der Punkt ist, dass durch die Einschränkungen der Wertebereiche der Variablen der Lösungsraum stark verkleinert wird. Anschließend reichen gute Heuristiken und die in Abschnitt 3.3.4 vorgestellte Strategie (ohne dieser war es niemals möglich ein Ergebnis zu erhalten!) rasch zu einer Lösung zu kommen. Probleme gibt es jedoch dann, wenn entweder keine Lösung gefunden werden kann aber kein „direkter“ Widerspruch aufscheint, oder es zu lange dauert, eine erste Lösung zu finden. Diese beiden Fälle können nicht auseinandergehalten werden. Der Solver bricht in beiden Fällen mit einer Fehlermeldung ab, und es gibt klarerweise keine verwertbare Lösung. Dann half es nur, mit den Daten so lange zu arbeiten, bis sich wieder eine Lösung finden ließ. Dazu zählt, Vorlesungen zu löschen, Einschränkungen aufzuheben und Verschränkungen zu lösen. Meinen Beobachtungen nach wirkten sich Einschränkungen in den Zeitspannen und den Überschneidungen von Vorlesungen nach deren thematischen Typen (siehe Abschnitt 3.3.3.4.4) besonders stark aus.

Ich war auch in Unterschieden zu den tatsächlich verwendeten Einteilungen interessiert. Meine grobe statistische Untersuchung zeigte, dass es durch die berechnete Einteilung zu einer viel stärkeren Verteilung der Vorlesungen kam. Da viele implizit angenommenen Einteilungen nicht angewendet wurden, kam es vor, dass relativ viele Seminare für den zweiten Abschnitt vormittags eingeteilt wurden. Das war eher untypisch, ließe sich jedoch mittels genauerer Einschränkungen nach Abschnitt 3.3.3.4.5 angeben, falls dies gewünscht ist. Genauso wurden einige Vorlesungen später angesetzt, als sie das gewöhnlich werden. Der Grund liegt meiner Meinung darin, dass die Bedingungen an die Überschneidungen der thematischen Typen von Vorlesungen eine starke Verteilung erzwingt. Was ein weiterer Grund sein könnte, ist die Einschränkung auf halbe Stunden Schritte bei der Zeitvariablen. Dies war nötig, um den Suchraum zu verkleinern um überhaupt eine Lösung zu erhalten. Da sich dies jedoch nicht exakt untersuchen lässt,

bleibt dies eine Vermutung und müsste genauer überprüft werden.

Eine andere Beobachtung war, dass von über 20 zufällig gewählten Wünsche mehr als die Hälfte erfüllt wurden. Dies hat mich etwas überrascht und bestärkt mich, dass es sich um einen guten Ansatz zur Behandlung der Wünsche handelt.

Daher lautet meine Schlussfolgerung, dass dieses Programm durchaus praktische Verwendung finden kann. Es sollte jedoch noch dahingehend verbessert werden, dass es eine Kombination mit lokalen Suchmethoden wie zum Beispiel der Tabu Suche aus Kapitel 1.2.3 für die thematischen Überschneidungen und Wünsche gibt. In diesem Fall müsste der ILOG Solver durch einen anderen ersetzt werden. Auch müssen noch Erfahrungen aus einem praktischen Einsatz gesammelt werden, um eventuell Änderungen am Modell vorzunehmen.

4 Handbuch zum Vorlesungsplaner

In diesem Abschnitt wird aus der Sicht eines Anwenders beschrieben, wie die Bedienung des Programmes konzipiert wurde. Es wird auf den gesamten Ablauf, die möglichen Einstellungen und unterschiedliche Szenarien eingegangen. Vor allem für den Administrator, der hauptsächlich für die Verwaltung der gesamten Daten zuständig ist, ist es unerlässlich, sich zuerst in die Konzepte einzuarbeiten, damit Fehlbedienungen keine vermeidbaren Fehler verursachen. Schon bei der Modellierung wurden unzählige Funktionen und Muster für Bedingungen bedacht, und folglich sollte für jede Verknüpfung oder Bedingung nur der genau dafür gewählte Weg gegangen werden, wenn mehrere denkbare Möglichkeiten zur Umsetzung vorhanden sind. Ansonsten könnten Seiteneffekte entstehen, die einen Mehraufwand für den Anwender bedeuten oder gar eine nicht gewünschte Einschränkung an das Gesamtsystem erzeugen.

Ein Beispiel wäre die Festlegung der Zeit einer Vorlesung auf den Vormittag. Dies ließe sich dadurch erreichen, dass alle Zeiten des Nachmittags verboten werden oder andererseits – positiv formuliert – nur die Zeiten des Vormittags als mögliche Zeiten zugelassen werden. Der zweite Weg ist der dafür vorgesehene, wohl aber gibt es für spezielle Fälle auch die erste Möglichkeit diese Information dem Programm mitzuteilen. Doch zuerst kommt ein systematischer Aufbau, wie der Vorlesungsplaner installiert, die Daten eingegeben werden und dann wie ein Vorlesungsplan erfolgreich berechnet wird.

4.1 Installation

Die Installation verläuft in mehreren Teilen und wird hier für ein von mir verwendetes Linux System erklärt. Ein verwandtes UNIX System wäre sicherlich ganz ähnlich zu behandeln. Für Windows kann ich keine genaueren Informationen geben.

Zuerst muss der ILOG Solver installiert werden. Dieser wird standardmäßig in `/usr/ilog/` entpackt und die Lizenzdatei installiert. Die Schnittstelle für die darauf basierende Software ist eine JNI Datei in: `/usr/ilog/OPLSt37/classes/opljni.jar`

Die Software für das Interface ist eine Webanwendung. Das ist deshalb notwendig, weil mehrere Anwender Daten eingeben und alles zentral verwaltet wird und einfach funktionieren soll. Auf diese Art ist nur ein Server für die gesamte Verwaltung und Applikationslogik notwendig – für die Anwender dann nur eine Netzwerkverbindung und ein Webbrowser. Daher muss nichts lokal installiert werden, und der Vorlesungsplaner ist plattformunabhängig nutzbar. Voraussetzung für die Webbrowser ist W3C konformes Verstehen von HTML mit CSS und JavaScript. Getestet auf korrektes funktionieren sind Mozilla Firefox 1.5, 2.0 oder höher und Microsoft Internet Explorer 6.0. Andere Webbrowser könnten wegen ungenügender JavaScript Implementation unvollständige Formulare liefern, da zur

Reduktion der zu übertragenden Formularseiten redundante Daten erst am Client generiert werden (mittels JavaScript und dem DOM-Modell der Webseiten). Das ist vor allem bei den sehr zahlreich auftretenden Auswahllisten notwendig.

Der Server selbst ist Apache Tomcat 5.5. Entwickelt wurde auf der Unterversion 5.5.9, aber alles sollte problemlos auch in einer aktuelleren Version 5.5.x funktionieren. Tomcat hat eine längere Geschichte, das ursprüngliche Programm wurde von Sun Microsystems entwickelt und dann später der Apache Foundation als Open-Source Programm übergeben. Es basiert auf Java und verwendet hier die offizielle Sun J2SE Version 1.5.0. Der Source Code des Programms ist in Level 1.5 geschrieben, entwickelt in Netbeans 5.0 unter Linux.

Es werden die folgenden zwei Open-Source Bibliotheken verwendet:

- dom4j: Bibliothek zur Verwaltung von XML Daten. Sie arbeitet sehr schnell, da sie die XML Daten direkt in Java-Datenobjekte abbildet und es lassen sich über XPath definierte Listen gut verarbeiten. [`www.dom4j.org`]
- iText: PDF Generator Bibliothek [`www.lowagie.com`]

Das Programm ist so konfiguriert, dass die Bibliotheken zusammen mit den Quellen verwaltet werden und Apache Ant über das entsprechende Build-Script die Bibliotheken in die für den Webserver Tomcat verwendete WAR-Distributionsdatei einbindet. Es muss also für den Server keine zusätzliche Vorkehrung für die Bibliotheken getroffen werden. Das Kompilieren läuft auf die Art, dass Apache Ant installiert sein muss und im Stammverzeichnis der Entwicklungsdateien folgender Befehl aufgerufen werden muss:

```
ant dist
```

Dann befindet sich die für den Tomcat Server notwendige WAR-Datei im `/dist/` Unterverzeichnis. Anschließend kann mit `ant clean` wieder aufgeräumt werden. Die JavaDoc-Dokumentation lässt sich mit `ant javadoc` erstellen. Die anderen Targets sind nur für die Entwicklung notwendig.

Allerdings muss der Pfad zu der über `JNI` eingebundenen ILOG-Solver Bibliothek angegeben werden. Dies muss in der `LD_LIBRARY_PATH` Umgebungsvariable für den Tomcat Server geschehen. Da es nicht zu empfehlen ist, den `LD_LIBRARY_PATH` systemweit zu setzen bzw. dieser oft durch andere Programme zurückgesetzt wird, sollte dies in dem Startup-Script von Apache Tomcat eingetragen werden. Nur so kann hunderprozentig sichergestellt werden, dass die Anbindung von Tomcat an den Solver funktioniert. Das entsprechende Start-Script befindet sich in

```
<installpath>/apache-tomcat-5.5.12/bin/
```

und heißt `start.sh`. Darin sollte man, am besten gleich am Anfang (vor oder nach dem Einleitungstext, das macht keinen Unterschied – jedoch nach der `#/bin/bash` Signatur des Skriptes in der ersten Zeile), folgende Zeile hinzufügen:

```
LD_LIBRARY_PATH=/usr/ilog/OPLSt37/lib/i86_linux2_glibc2.3_gcc3.2/shared/
```

Nur so kann der Tomcat Server auf den ILOG-Solver zugreifen. Ist der ILOG-Solver in einem anderen Verzeichnis als `/usr/ilog/` installiert, muss die Zeile klarerweise angepasst oder ein Symlink erstellt werden.

Die weitere Installation läuft so, dass die mit Apache Ant kompilierte WAR-Datei über das Managementinterface des Tomcat Servers hochgeladen wird und dann startet. Dann ist über `<server> /VOPlaner` die Anwendung erreichbar. Falls notwendig, lässt sich das Unterverzeichnis über die Konfigurationsdateien des entpackten Programmes anpassen. Um wirklich lauffähig zu sein, müssen weitere Dateien vorhanden sein.

4.2 Arbeits- und Konfigurationsdateien

Das Programm ist so aufgebaut, dass über die Webmaske die Daten zur Berechnung des Modells gesammelt werden. Diese werden in einer XML Datei gespeichert. Dadurch bleibt die Verarbeitung der Daten flexibler als mit einer Datenbank und ist nicht an die Installation einer Datenbank gebunden. Es kann bei abgeschaltetem Service die XML Datei mit einem beliebigen Texteditor bearbeitet werden. Es muss nur aufgepasst werden, dass die Integrität des Zeichensatzes und der Daten erhalten bleibt. Außerdem können durch Kopieren der Datei jederzeit auf einfachste Weise Backups erstellt werden, bzw. ältere Daten die aktuelle ersetzen, um eine andere Konfiguration zu bearbeiten.

Eine weitere Konfigurationsdatei für die Passwörter der Zugriffskontrolle ist ebenfalls vorhanden. Sie wird genauso wie die für die Daten behandelt. Ihr Zweck ist einzig das Speichern der Passwörter und der Rechte. Es können theoretisch auch weitere Konfigurationen für das Programm abgelegt werden, welche hier unabhängig von den Daten verwaltet werden.

Der Ort, an dem diese XML-Dateien gespeichert werden, muss in den Umgebungsparametern des Tomcat Servers angegeben werden. Diese können über die Administrationsschnittstelle bearbeitet werden. Direkt ist dies auch in der `web.xml` Datei des installierten Services möglich.

4.3 Bedienung

Es folgt eine Erklärung, wie das Programm benützt werden soll. Derjenige, der das Programm administriert, muss sich mit den einzelnen Vorgängen vertraut machen und sollte entsprechend den hier vorgestellten Anweisungen handeln. Das ist deshalb notwendig, weil viele dieser Schritte voneinander abhängen und bei dem Entwurf des Programms nur an eine bestimmte Bedienweise gedacht wurde. Beispielsweise lassen sich Vorlesungen erst dann eintragen, wenn die jeweils dazugehörigen Daten des Vortragenden eingegeben sind.

4.3.1 Schritt 1 - Basiskonfiguration

Im Vorfeld eines neuen Stundenplanes muss ein neues leeres XML Dokument für die Daten vorhanden sein. In ihm sind grundlegende Parameter und Variablen gespeichert.

Dies beinhaltet die Anzahl und Bezeichnung der einzelnen Tage, die Typen der Orte, die thematischen Typen der Vorlesungen (Fachgruppen und Abschnitte des Studiums) und ähnliches und muss bei Bedarf manuell in der XML Datei angepasst werden. Standardmäßig ist jedoch schon eine für die meisten Fälle ausreichende Konfiguration vorgegeben. Wichtig ist zu wissen, dass diese Werte nachträglich nicht geändert werden können. Dies ist eine generelle Einschränkung, damit auf jeden Fall sichergestellt wird, dass die Daten konsistent bleiben, weil es viele direkte und indirekte Abhängigkeiten gibt. Eine Implementation von Methoden um diese Daten nachträglich zu ändern wäre ein sehr großer Aufwand bei gleichzeitig geringem Nutzen. Alle wichtigen Änderungen können ohnehin nachträglich gemacht werden, und als Faustregel gilt, dass die Daten, welche von einem Semester auf das nächste konstant bleiben, fix sind – hingegen alles was sich von einem Semester auf das Nächste ändern kann, sich über die Webschnittstelle anpassen lässt.

4.3.2 Schritt 2 - Dateneingabe

Bevor das Modell berechnet werden kann, müssen die dafür notwendigen Daten eingegeben werden. Diese teilen sich grundsätzlich in zwei Teile: erstens die für Hörsäle, Vortragende und die Vorlesungen und zweitens die Daten, welche Beziehungen zwischen den Vorlesungen angeben oder allgemeine Randbedingungen. Hier geht es um ersteren Typ und die dafür notwendigen Grundlagen.

4.3.2.1 Grundlegendes

Bevor die Hörsäle, Vortragenden und Vorlesungen eingetragen werden können, müssen die Tage der Woche vorgegeben werden. Das kann nur, wie in der Einleitung beschrieben, direkt in der XML Datei erfolgen. Darüber hinaus müssen die Zeitspannen angegeben werden. Das geschieht über die Webmaske und sollte im Voraus durchgeführt werden. Es kann zu unerwünschten Nebeneffekten kommen, wenn hier nachträglich Veränderungen vorgenommen werden, da die entsprechenden Werte in den anderen Daten mitverändert werden, um die Konsistenz zu erhalten, und außerdem leere Datenfelder nicht erlaubt sind. Dies ist bei Datenbanken eine übliche Vorgangsweise, und dieses Verhalten wird *on update cascade* genannt. Deshalb könnten schnell viele Datenwerte verändert worden sein, ohne dass dies explizit gewünscht wurde.

Die Hauptdaten sollten in der hier besprochenen Reihenfolge eingegeben werden. Es ist aber immer möglich, nachträglich da und dort Ergänzungen und Modifikationen zu machen. Man sollte dabei allerdings bedenken, dies gleichzeitig nur jeweils einzeln zu machen, weil es untereinander Verknüpfungen gibt. Es wird zwar immer überprüft, ob die Daten übernommen werden können, stört aber die Eingabe, wenn Daten aus Konsistenzgründen nicht angenommen werden dürfen.

Bei der Eingabe von Identifikationen ist darauf zu achten, keine Sonderzeichen, Umlaute und Leerzeichen zu verwenden. Dies ist deshalb notwendig, weil diese Schlüsselwörter vom ILOG Solver direkt verwendet werden. Dieser kann nur mit einem begrenzten Zeichensatz arbeiten. Es wäre auch möglich gewesen hier, neue Identifikationen zu erfinden und Sonderzeichen zuzulassen, jedoch ist dann die generierte Daten-Datei für den Solver

nicht vom Menschen lesbar. In der Planung dieses Programmes war es mir jedoch wichtig, an so vielen Stellen wie möglich die Lesbarkeit der Textdateien für den Menschen zu gewährleisten. Für besondere Zwecke lassen sie sich dann später noch händisch weiter bearbeiten.

4.3.2.2 Hörsäle

Hörsäle sind mit einer eindeutigen Identifikation versehen, am besten der Bezeichnung des Raumes. Die maximale Anzahl Zuhörer muss ebenfalls eingegeben werden. Die Zeiten, an denen der Hörsaal zur Verfügung steht, wird standardmäßig auf alle Tage gesetzt – kann jedoch auch weniger sein. Die Zeitspannen, die davon noch zusätzlich ausgeschlossen werden, sind nach der Eingabe anschließend einzugeben. Letzteres eignet sich dafür, Hörsaalbelegungen durch andere vorzugeben.

4.3.2.3 Vortragende

Die Vortragenden werden ähnlich wie die Hörsäle eingegeben. Sie haben eine eindeutige Identifikation und stehen ebenfalls immer zur Verfügung, außer wenn sie entweder nicht den ganzen Tag Zeit haben oder Zeitspannen sie ausschließen. Diese Informationen werden im Bereich für die Vortragenden eingegeben. Der Administrator gibt den Vortragenden nur ihre Identifikation und die Gewichtung. Diese Gewichtung ist ein rationaler Wert zwischen 0 und 1, welcher linear als Gewichtung in die Zielfunktion der Optimierung einfließt. Eine höhere Gewichtung macht den Wunsch an die Vorlesung eines Vortragenden bedeutender.

Anschließend müssen die Meta-Vortragenden, das sind virtuelle Gruppen von mehreren Vortragenden, die gemeinsam eine Vorlesung halten, angegeben werden. Erst wenn sie definiert sind, stehen sie bei der Eingabe dieser Gruppenvorlesungen zur Verfügung.

4.3.2.4 Vorlesungen

Vorlesungen sind komplizierter als die vorhergehenden Datenstrukturen. Abbildung 4.1 (Seite 90) zeigt, wie die Benutzerschnittstelle aussieht. Nebst einer jeweils eindeutigen Identifikation müssen noch eine Reihe weitere Informationen angegeben werden. Es muss angegeben werden von wem die Vorlesung gehalten wird. Das kann ein einzelner Vortragender oder ein Meta-Vortragender sein. Außerdem müssen die Zeitdauern der einzelnen Teile der Vorlesung durch Leerzeichen getrennt angegeben werden. Besteht sie nur aus einem Teil, so ist natürlich nur ein Wert anzugeben. Die Angabe erfolgt in Minuten. Dadurch lassen sich auch ungerade Teilungen von dreistündigen Vorlesungen, die 150 Minuten dauern, in zwei ungleich große Teile wie 60 und 90 eingeben.

Der zeitliche Typ muss angegeben werden, oder der allgemeine Typ, der keine Einschränkung bedeutet. Dies ist zur Unterscheidung von tagsüber und abends gehaltenen Vorlesungen notwendig. Außerdem gibt es feinere Kategorien für in der früh oder nachmittags gehaltenen. Die Typen müssen im Vorfeld angegeben werden, die genaue zeitliche Einteilung kann jederzeit über die Webmaske editiert werden. Hier ist nur eine Referenz

auf diese Zeitspanne angegeben. Das bedeutet, dass sich nachträglich die zur Verfügung stehenden Zeitspannen für viele Vorlesungen leicht gleichzeitig ändern lassen.

Der inhaltliche Typ der Vorlesung ist die wahrscheinlich stärkste Randbedingung der Vorlesungen. Er beschreibt, welche Vorlesungen sich mit welchen anderen nicht überschneiden dürfen. Dies sind zum Beispiel die Hauptvorlesungen gegen deren Proseminare. Jedoch ist es kein Problem, wenn sich die einzelnen Teile der Proseminare untereinander überschneiden, da es sich um separate Gruppen handelt. Außerdem kann hier verhindert werden, dass eine für den zweiten Abschnitt verpflichtende Vorlesung in zeitlicher Konkurrenz zu einer der speziellen Vorlesungen des zweiten Abschnitts angeboten werden. Konkret für den Fall des Mathematikinstitutes muss teilweise sogar noch feiner eingeteilt werden, da es sonst zu zu vielen ausschließenden Bedingungen kommt.

Ein spezieller Typ von Überschneidung betrifft das PC-Labor. Das wird hier so gehandhabt, dass bei aktivieren dieses Feldes sich keine der Vorlesungen überschneiden darf. Das ist, was die Praxis betrifft, vermutlich ebenfalls sehr einschränkend und sollte mit Vorsicht benützt werden. Realistisch ist es immer möglich, das PC-Labor zu besuchen, da es sich um deren drei handelt und hier nur eine allgemeine Wahlmöglichkeit besteht. In der detaillierten Diskussion des Modells wird darauf genauer eingegangen und die damit verbundenen Probleme und Lösungen besprochen.

Viel wichtiger ist die Wahl der Verknüpfung der einzelnen Teile einer Vorlesung untereinander. Traditionell ist es häufig so, dass mehrteilige Vorlesungen zur selben Zeit im selben Hörsaal aber an verschiedenen Tagen gehalten werden. Deshalb muss dies durch das Optionsfeld “TZSZ“ (*Teile Zur Selben Zeit*) angegeben werden. Ebenfalls unbedingt notwendig ist die Option „TAAST“ (*Teile Auch Am Selben Tag*) für Proseminare. Vorallem bei Anfängervorlesungen gibt es mehr Teile als Tage der Woche, und da normalerweise implizit vorgegeben ist, dass keine zwei Teile einer Vorlesung am selben Tag gehalten werden, würde dies sofort die Unlösbarkeit zur Folge haben. Die Option “TA“ (*Teile Aufsteigend*) besagt, dass die Vorlesungsteile exakt in der hier angegeben Reihenfolge eingeteilt werden müssen. Dies hat viele Vorteile und sollte unbedingt auch bei den Vorlesungen gemacht werden, die nur aus gleich langen Teilen bestehen. Dies schränkt die Wahlfreiheit für den Lösungsalgorithmus ein, ohne dabei aus Symmetriegründen eine mögliche Lösung zu eliminieren. Im häufigsten Normalfall ist es also notwendig, “TZSZ“ und “TA“ auszuwählen.

4.3.2.5 Restliche Daten

Dazu zählen die Informationen der einzelnen Vortragenden über ihre Anwesenheit. Diese wird nicht über das Administrationsaccount sondern in dem für die Vortragenden eingegeben.

4.3.3 Schritt 3 - Bedingungen an den Vorlesungsplan

Die Daten über die genaue Einteilung der Vorlesungen und die Verknüpfungen untereinander sind entscheidend, ob ein berechnetes Modell auch wirklich den geforderten Wünschen an die Lösung entspricht. Es sind diverse Möglichkeiten vorhanden, detailliert in die berechnete Einteilung einzugreifen. Es folgt nun eine Auflistung der möglichen

Randbedingungen, thematisch gruppiert und in einer Reihenfolge, die für die Eingabe der Daten nützlich ist.

4.3.3.1 Fixe Vorgaben

Zuerst müssen die direkten Randbedingungen an alle Daten der Hörsäle, Vortragenden und Vorlesungen eingegeben werden. Dies beinhaltet die Zeitspannen, an denen sie zur Verfügung stehen und die davon ausgeschlossenen Zeitspannen. Diese Daten sollten so gewählt werden, dass nur geringe Einschränkungen an den gesamten Lösungsraum gemacht werden. Was die Vorlesungen betrifft, gibt es mehrere Möglichkeiten. Sollen sie zu bestimmten Zeitspannen wie abends oder vormittags gehalten werden, so wird das direkt in den Daten zu jeder Vorlesung angegeben. Wird gewünscht, dass sie darüber hinaus an einem bestimmten Tag oder zu einem genauen Zeitpunkt gehalten werden sollen, so wird dies in den verpflichtend gültigen Bedingungen eingetragen. Letzteres ist für Vorlesungen notwendig, die schon im Vorfeld fix eingeteilt werden. Genau diese sollten auch zuerst eingetragen werden. Das hat den Hintergrund, dass bereits zu diesem Zeitpunkt die Daten auf Konsistenz geprüft werden können. Das geschieht so, dass testweise ein Vorlesungsplan berechnet wird. Gibt es eine Lösung, sind diese fix eingegebenen Daten konsistent.

Tests und praktische Versuche im Verlauf der Entwicklung dieser Software zeigten, dass hier leicht die zeitlichen Beschränkungen der Vortragenden gegenüber den festgelegten Zeiten der Vorlesungen in Widerspruch stehen und genau kontrolliert werden müssen. Solche sich ausschließende Situationen können leicht eintreten und können in vielfältigster Art auftreten.

4.3.3.2 Notwendige Randbedingungen

Zusätzlich zu den fix vorgegebenen Zeiten für die Vorlesungen gibt es noch eine Reihe anderer Bedingungen, die in den Vorlesungsplan einfließen können. Dabei handelt es sich um Pausen bei bestimmten Situationen zwischen den Vorlesungen, Verknüpfungen zwischen zwei Vorlesungen und die thematische Charakterisierung der Vorlesungen zu Gruppen, um ihre Überschneidungen untereinander zu ermöglichen oder zu verhindern. Diese Typisierung der Vorlesungen mit der dazugehörigen Überschneidungsmatrix ist in Abbildung 4.2 (Seite 91) zu sehen.

Es lässt sich dabei festlegen, ob zwei Teile einer Vorlesung am selben Tag gehalten werden müssen oder im Gegenteil nicht am selben Tag gehalten werden dürfen. Außer dem Tag lassen sich auch die Zeit und der Hörsaal synchronisieren oder erzwungenermaßen ungleich machen. Bei der Zeit bedeutet dies, dass nicht nur die Beginnzeiten ungleich sind, sondern sich die Vorlesungen nicht überlappen dürfen und es sich damit genau gleich wie mit den Zeittypen verhält.

4.3.4 Schritt 4 - Wünsche

Zusätzlich zu diesen Zwangsbedingungen ist es außerdem möglich, nicht unbedingt zu erfüllende Bedingungen anzugeben. Diese Wünsche werden, je nach der Gewichtung des dazugehörigen Vortragenden, in ihrer Priorität gereiht. Es ist möglich, Wünsche für die Teile einer Vorlesung für bestimmte Tage, einen Zeitpunkt oder Hörsaal zu wählen. Werden mehrere Wünsche eingegeben, so wird versucht aus dieser Liste einen dieser Wünsche zu wählen. Dabei sind Tag- und Zeit-Wünsche miteinander gekoppelt, Hörsaalwünsche von ihnen unabhängig. Entsprechend lässt sich zum Beispiel der Wunsch abgeben, einen Tag auszuschließen und gleichzeitig einen bestimmten Hörsaal zu wählen. Intern wird danach getrachtet, beide Wünsche oder nur einen von beiden, zu erfüllen und es verfällt nicht der eine Wunsch, wenn der andere nicht erfüllbar ist.

4.3.5 Schritt 5 - Lösung und Ausgabe

Die Berechnung und Ausgabe einer Lösung sind zwei separate Schritte. Das liegt nicht nur daran, dass beide Vorgänge Rechenzeit benötigen, sondern sie auch separat gehandhabt werden sollten. Es soll also nicht jedes mal beim Abfragen einer Darstellung der Lösung das Modell erneut gelöst werden, sondern hier auf die letzte für in Ordnung befundene Lösung zurückgegriffen werden. Wird aber eine Lösung berechnet, so wird die alte durch diese neue ersetzt – auch dann, wenn keine Lösung gefunden werden konnte.

Das Auffinden der Lösung ist nur für den Administrator möglich. Es wird direkt die Berechnung durchgeführt, was durch eine in Echtzeit aktualisierte Ausgabe im Fenster des Browsers dokumentiert wird (Abbildung 4.3 auf Seite 92). Es muss darauf geachtet werden, nur einen Lösungsvorgang zur selben Zeit zu starten, weil konkurrierende Lösungsschritte keinen Sinn machen. Die der Lösung zugrunde liegenden Daten werden bei jedem neuen Lösungsversuch aus den in diesem Augenblick aktuellen erstellt. Diese sind separat abrufbar – das ist die .dat-Datei – und könnte auch direkt im ILOG-Solver verwendet werden.

Die Ausgabe kann auf zwei Arten erfolgen: als HTML-Datei oder PDF Dokument. Es wird auf Basis der zum damaligen Zeitpunkt der Berechnung der Lösung aktuellen Daten und der Lösung selbst eine übersichtliche Zusammenfassung erstellt. Beide sind nur so lange aktuell wie es die Lösung gibt. Wird sie durch eine neu berechnete ersetzt, so sind sie verfallen. Werden sie später für Vergleichszwecke gebraucht, müssen sie separat gespeichert werden. Ein Beispiel (die Ergebnisse sind nur auszugsweise zu sehen) ist in Abbildung 4.4 zu sehen. PDF und HTML Output sind vollkommen gleich.

4.3.5.1 Auffinden von Widersprüchen

Kann der Solver keine Lösung finden, so gibt es ein Hilfsmodul, um auf mögliche Widersprüche zu stoßen. Da es aber im Allgemeinen sehr kompliziert ist, das zugrunde liegende Problem zu finden, ist diese Methode oft nicht zielführend. Tests haben gezeigt, dass es nur zwei Strategien überhaupt geschafft haben, einfache Widersprüche zu finden. Die eine Methode versucht durch obere Schranken an die Maximalzahl an Verletzungen an die

Randbedingungen und einer geschickten Minimierung ihrer Anzahl genau die problematischen Bedingungen einzugrenzen. Die andere Methode funktioniert ganz ähnlich, jedoch versucht sie etwas allgemeinere Verletzungen zuzulassen. Dadurch steigt die Erfolgsaussicht eine Aussage über die Unlösbarkeit zu finden, jedoch ist diese von viel allgemeinerem Charakter und daher weniger aussagekräftig. Ausgegeben werden immer Vorlesungsteile und dazugehörige Arten von Randbedingungen. Es sollte dabei aber bedacht werden, dass das Programm keinerlei Wissen über den Charakter der Randbedingungen hat, da es keine semantischen Informationen gibt. Daher sind die durch diese Prozedur gewonnen Informationen nicht unbedingt die, welche auch wirklich den Kern des Problems darstellen. Eine genaue Betrachtung ist auf jedenfall notwendig. Weiters können nur wenige Widersprüche gefunden werden, da ansonsten der Raum der Wahlfreiheiten für den Solver zu groß wird. Es ist daher ratsam, während der Eingabe der Daten immer wieder den Solver laufen zu lassen, um schon früh Widersprüche finden zu können.

4.3.6 Schritt 6 - Backup

Die gesamten eingegeben Daten werden in einer einzigen XML-Datei gespeichert. Diese lässt sich für den Administrator einsehen und über den Webbrowser herunterladen oder direkt im Dateisystem darauf zugreifen. In beiden Fällen muss diese Datei nur unter einem aussagekräftigen Namen gespeichert werden und dient dann als Snapshot des aktuellen Zustandes und kann später wiederverwendet werden. Es wird empfohlen, regelmäßig Backups der Daten zu machen, da es nicht ausgeschlossen ist, dass es zu Fehlern oder Inkonsistenzen kommt.

Der Restore Prozess sieht so aus, dass das gesamte Service gestoppt werden sollte und dann entweder im Setup des Services die Backupdatei als aktuelle angegeben wird oder die Arbeitsdatei durch die Backupdatei überschrieben wird. Wird das Service wieder gestartet, wird automatisch ein allgemeiner Test auf Gültigkeit der XML-Datei gemacht und sie dann als Basis für alle Operationen verwendet.

4.3.7 F. A. Q.

Die folgenden Fragen und Antworten stellen mögliche Anforderungen an die Lösung vor und die anschließende Antwort erklärt den empfohlenen Weg, sie dem Programm anzugeben. Kann ein angegebener Wunsch nicht erfüllt werden, so wird nicht nur darauf hingewiesen, dass dies nicht möglich ist, sondern eventuell auch eine ähnliche Möglichkeit angegeben den Wunsch anders oder nur teilweise zu erfüllen.

Wie kann sichergestellt werden, dass eine Vorlesung innerhalb einer bestimmten Zeitspanne gehalten wird?

Es muss bei den Eigenschaften der Vorlesung bzw. deren Teile die Zeitspanne ausgewählt werden. Es müssen im Voraus die benötigten Zeitspannen angegeben werden, die dann bei mehreren Vorlesungen verwendet werden können. Achtung: Nachträgliche Änderungen an den Zeitspannen gelten für alle entsprechenden Vorlesungen.

Wie wird festgelegt, dass unterschiedliche Teile einer Vorlesung an verschiedenen Tagen gehalten werden?

Standardmäßig werden sie auf unterschiedliche Tage eingeteilt – dies kann durch die Option `TAAST` unterbunden werden.

Die Teile einer Vorlesung sollen in ihrer angegeben Reihenfolge sortiert bleiben!

Es muss bei den Einstellungen für die Vorlesung die Option `aufsteigend` angegeben werden.

Zwei Vorlesungen bzw. ihre Teile und eine andere Vorlesung bzw einer ihrer Teile sollen aufeinanderfolgend sein – d.h. später am selben Tag oder an einem darauf folgenden.

Dies kann durch die `später`–Relation angegeben werden. Achtung: Besteht eine Vorlesung aus mehreren Teilen und sollen alle Teile nach einer anderen Vorlesung stattfinden, so reicht es nicht, nur den ersten Teil dieser Vorlesung nach der anderen zu setzen. Es muss auch festgelegt werden, dass alle Teile nach dem ersten Teil stattfinden.

Wie lässt sich dies zusätzlich nicht nur am selben Tag, sondern für den darauf folgenden erreichen?

Hier muss zusätzlich angegeben werden, dass diese beiden Teile nicht am selben Tag stattfinden dürfen. Zusammen heißt das, dass die andere Vorlesung sicher auf den vorhergehenden Tag gesetzt sein wird.

Wo kann ich angeben, dass sich zwei Vorlesungen *nicht* überschneiden dürfen?

Das kann auf zwei Arten geschehen. Entweder mit der Überschneidungsmatrix für die Vorlesungstypen. Das ist die allgemeine Art. Handelt es sich jedoch um zwei Vorlesungen, die sich unabhängig davon nicht überschneiden dürfen, so kann dies über die *selbe Zeit* Bedingung (siehe Seite 55) geschehen. Es muss bei der Zeit die `-1` für *nicht* gewählt werden.

Wie kann ich Wünsche an einen bestimmten Termin angeben?

Das kann im Bereich der *Professoren* eingegeben werden, und diese Wünsche werden nach den Gewichtungen der Vortragenden bewertet. Diese Wünsche sind allerdings nicht verpflichtend – dies muss der Administrator bei den verpflichtenden Angaben für Vorlesungen festlegen.

Wie lässt sich eine bestimmte Zeitspanne für die Verfügbarkeit eines Hörsaales ausschließen?

Bei den Einstellungen zu jedem Hörsaal kann nicht nur die Zeitspanne zu der er pro Tag zur Verfügung steht angegeben werden, sondern es können auch Zeitspannen ausgeschlossen werden.

Wie lässt sich für die Verfügbarkeit eines Vortragenden eine bestimmte Zeitspanne ausschließen?

In den Einstellungen, die im Bereich der *Professoren* verfügbar sind, können die pro Vortragenden verfügbaren Zeitspannen und Ausschlusszeiten angegeben werden. Hier sollte nicht alles bis auf einige wenige Wunschzeiten ausgeschlossen werden, da es sonst praktisch unmöglich wird, eine Lösung zu finden. Wünsche sollten nur als Wünsche angegeben werden und hier so viele Zeitbereiche wie möglich angegeben werden.

Es soll eine Vorlesung erst vor/nach einem bestimmten Tag gehalten werden.

Es sind keine $\leq$ oder $\geq$ Bedingungen mit Tagen vorgesehen. Nur zwei Vorlesungen zueinander können zeitlich geordnet werden. Als Alternative können Tage mit *Pflichten* ausgeschlossen werden.

Wie erreiche ich es, dass eine Vorlesung zu einem genauen Zeitpunkt oder Hörsaal eingeteilt wird?

Das geht als Administrator über die Pflichten. Es können Tag, Zeit oder/und Ort festgesetzt werden. Es sollte aber darauf hingewiesen werden, dass durch die Zeitbereiche jeder Vorlesung diese schon automatisch innerhalb einer bestimmten Zeitspanne gesetzt werden und aufgrund der angegebenen Anzahl an Zuhörern die Wahl des Hörsaals eingeschränkt wird. Oft sind diese Einschränkungen schon ausreichend.

Wie erzwinge ich, dass mehrere Lehrveranstaltungen am selben Tag gehalten werden?

Das kann über die Koppelungsbedingungen angegeben werden. Das kann derselbe Tag, dieselbe Zeit oder/und derselbe Hörsaal sein. Sollen mehrere Lehrveranstaltungen gekoppelt werden, so muss dies in einer Kette von Paaren angegeben werden. Aufzupassen ist bei Teilen derselben Vorlesung, da hier auch die Option `TAAST` angegeben werden muss, welche es dem Solver gestattet, Teile einer Vorlesung auch am selben Tag einzuteilen.

Es sollen zu einer Vorlesung mehrere Vortragende Zeit haben. Wie lässt sich sicherstellen, dass es zu keinen Überschneidungen kommt?

Hierfür sind die künstlich eingeführten *Meta-Professoren* notwendig. Diese stellen sicher, dass sie alle anwesend sein können. Ist aber nur die Anwesenheit von mindestens einem notwendig, so wird empfohlen, darauf zu verzichten und diesen ganz normal als einzigen Vortragenden für diese Vorlesung einzutragen. Es ist nicht vorgesehen, feinere Einteilungsmöglichkeiten vornehmen zu können.

Wie funktioniert die Verteilung von Vorlesungen in räumlich getrennten Gebäudekomplexe?

Liegen zwei Gebäude so weit auseinander, dass die Wegzeit relevant wird, so kann dies in der Matrix der Ortsgruppen eingetragen werden. Dort sind die Wegzeiten eingetragen

und sie wird über diese Örtlichkeiten indiziert. Dies hat nicht nur den Sinn, verschiedene thematisch getrennte Bereiche von mehreren Hörsälen zu trennen, sondern auch genau diese Wegzeiten miteinzubeziehen. Dies ist dann relevant, wenn ein und derselbe Vortragende an zwei unmittelbar aufeinander folgenden Terminen eine Vorlesung halten muss aber dies aufgrund der dazwischenliegenden Wegstrecke nicht schaffen kann. Hiermit liegt also ein sehr allgemein gefasstes Modell zugrunde, das theoretisch unbegrenzt große Areale umspannen kann. Darüber hinaus muss zu jeder Vorlesung angegeben werden, aus welchen Ortsbereichen der für sie passende Hörsaal gewählt werden muss.

In der Praxis heißt das zum Beispiel, dass eine Einführungsvorlesung für Physik in einem der Hörsäle der Strudlhofgasse gehalten werden muss, während hingegen eine Einführungsvorlesung für Mathematik in der Althanstraße gehalten wird, aber nicht in der Strudelhofgasse. Ist darüber hinaus derselbe Vortragende vorgesehen, so kann er aber nicht sofort zwischen beiden Bereichen seinen Standort wechseln, und daher muss zwischen diesen Einführungsvorlesungen mindestens die angegebene Zeitspanne Pause sein.

Die zwei Proseminare zu einer Vorlesung oder zwei Vorlesungen an sich sollen nicht am selben Tag stattfinden. Wie lässt sich dies angeben?

Es gibt die Bedingungen für Paare von Vorlesungen. Diese können auch ausschließendes angeben, d.h. dass der Tag ungleich sein muss. Dies ist dann günstig, um die beiden Termine für ein Proseminar nicht am selben Tag zu haben, auch wenn die für Proseminare empfohlene Option “TAAST“ angegeben wurde. Gibt es aber nur zwei Proseminare, so kann sowohl die “TAAST“ als auch diese ausschließende Bedingung weggelassen werden, um dieselbe Wirkung zu haben. Dies empfiehlt sich vor allem für die Proseminare mit vielen einzelnen Terminen weil so eine eventuelle Häufung an einem Tag vermieden wird, was in der Praxis nicht besonders günstig ist.

Ist es möglich, mehrere Zeit-Typen für eine Vorlesung zu wählen?

Nein, es lassen sich aber weitere Typen einrichten, die diese größere Zeitspanne abdecken.

Wie lässt sich verhindern, dass eine der verpflichtenden Vorlesungen des zweiten Abschnittes mit einer Vorlesung eines Studienzweiges kollidiert?

Dies und ähnliches wird ganz allgemein über eine Überschneidungsmatrix geregelt. Das ist eine symmetrische Matrix, deren Einträge binäre Werte sind. Ist der Wert eines Eintrages wahr, so dürfen sich die dem Eintrag entsprechenden Typen nicht überschneiden. Freilich ist es hierfür notwendig, genügend viele Typen bereitzustellen und die Matrix gut durchdacht anzugeben.

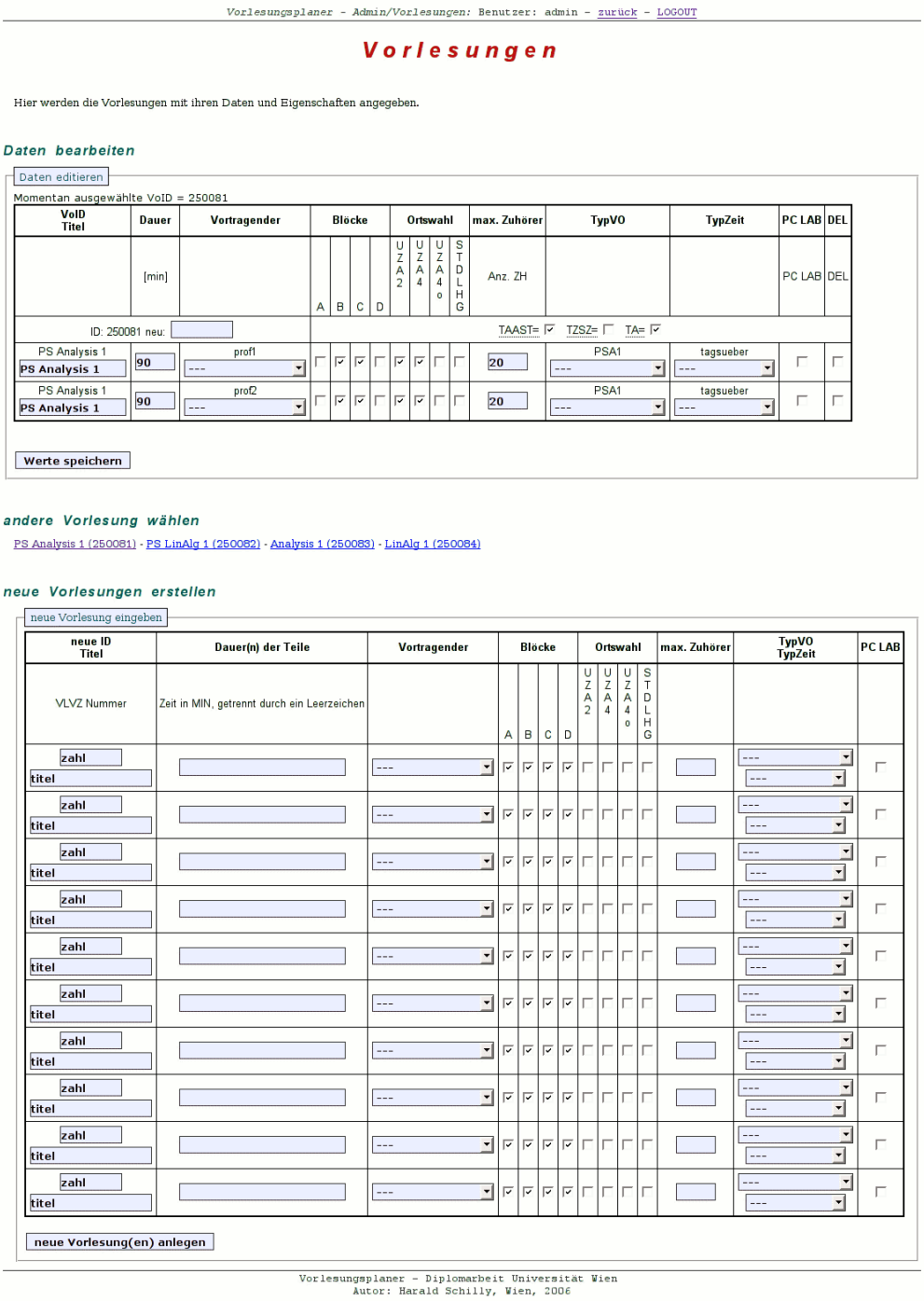
Zwei Vorlesungen werden wochenweise abwechselnd gehalten. Wie lässt sich das angeben?

Hierfür müssen zwei verschiedene Blöcke verwendet werden. In dem einen Block sind alle Vorlesungen und die erste, im zweiten sind auch alle und die zweite Vorlesung. Dadurch

kann über die Pflichtbedingungen auch problemlos angegeben werden, dass diese beiden abwechselnd gehaltenen Vorlesungen im selben Hörsaal am selben Tag und der selben Zeit gehalten werden müssen, ohne dass es ein Problem mit der scheinbaren Überschneidung gibt.

Lässt sich ein gemeinsamer Termin für die Vortragenden einrichten, an denen keiner von ihnen eine Vorlesung halten muss?

Dies ist direkt nicht vorgesehen, jedoch indirekt kann es auf die Art angegeben werden, dass diese Besprechungszeit als Vorlesung eingetragen wird und alle Vortragenden hierfür als “Meta“-Vortragende eingetragen werden. Als Hörsaal muss das Besprechungszimmer eingetragen werden. Alternativ lässt sich auch eine fixe Zeitspanne bei jedem Vortragenden angeben, zu der er keine Zeit hat. Letzteres ist sicherlich die für die Praxis günstigere Art.



Vorlesungsplaner - Admin/Vorlesungen: Benutzer: admin - zurück - LOGOUT

Vorlesungen

Hier werden die Vorlesungen mit ihren Daten und Eigenschaften angegeben.

Daten bearbeiten

Daten editieren

Momentan ausgewählte VoID = 250081

VoID Titel	Dauer	Vortragender	Blöcke A	B	C	D	Ortswahl UZA2	UZA4	UZA4o	STDLHG	max. Zuhörer	TypVO	TypZeit	PC LAB	DEL
	[min]										Anz. ZH			PC LAB	DEL
ID: 250081 neu:			TAAST= ☑ TZSZ= ☐ TA= ☑												
PS Analysis 1 PS Analysis 1	90	prof1 ---	☐	☑	☑	☐	☑	☑	☐	☐	20	PSA1 ---	tagsueber ---	☐	☐
PS Analysis 1 PS Analysis 1	90	prof2 ---	☐	☑	☑	☐	☑	☑	☐	☐	20	PSA1 ---	tagsueber ---	☐	☐

Werte speichern

andere Vorlesung wählen

PS Analysis 1 (250081) - PS LinAlg 1 (250082) - Analysis 1 (250083) - LinAlg 1 (250084)

neue Vorlesungen erstellen

neue Vorlesung eingeben

neue ID Titel	Dauer(n) der Teile	Vortragender	Blöcke A	B	C	D	Ortswahl UZA2	UZA4	UZA4o	STDLHG	max. Zuhörer	TypVO TypZeit	PC LAB
VLVZ Nummer	Zeit in MIN, getrennt durch ein Leerzeichen		☑	☑	☑	☑	☐	☐	☐	☐		--- ---	☐
zahl titel		---	☑	☑	☑	☑	☐	☐	☐	☐		--- ---	☐
zahl titel		---	☑	☑	☑	☑	☐	☐	☐	☐		--- ---	☐
zahl titel		---	☑	☑	☑	☑	☐	☐	☐	☐		--- ---	☐
zahl titel		---	☑	☑	☑	☑	☐	☐	☐	☐		--- ---	☐
zahl titel		---	☑	☑	☑	☑	☐	☐	☐	☐		--- ---	☐
zahl titel		---	☑	☑	☑	☑	☐	☐	☐	☐		--- ---	☐
zahl titel		---	☑	☑	☑	☑	☐	☐	☐	☐		--- ---	☐
zahl titel		---	☑	☑	☑	☑	☐	☐	☐	☐		--- ---	☐
zahl titel		---	☑	☑	☑	☑	☐	☐	☐	☐		--- ---	☐

neue Vorlesung(en) anlegen

Vorlesungsplaner - Diplomarbeit Universität Wien
Autor: Harald Schilly, Wien, 2006

Abbildung 4.1: Screenshot – Editieren der Vorlesungen

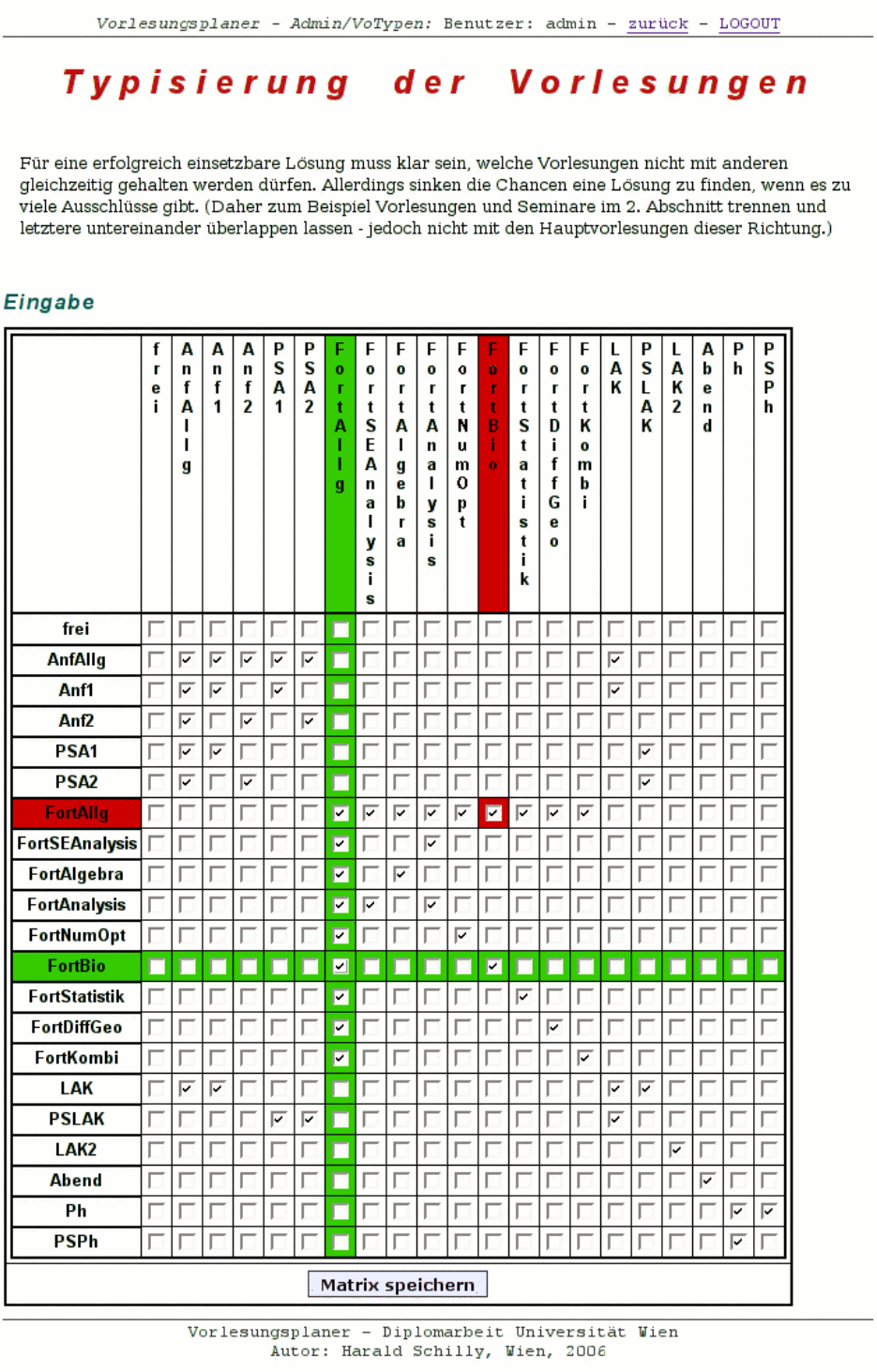


Abbildung 4.2: Screenshot – Überschneidungsmatrix der Vorlesungstypen

```
thld_MAX==2
thld_CNT==2
SOLVER:
initialisieren der opl solver engine

initialisieren und starten des keep connection alive threads

initialisieren des error reporters

error reporter aktivieren

laden der .mod datei + .dat datei
variables = 468 // constraints = 660

solver pass: 1
optimal solution? -> 0
time = 0.03200199999999942
objective value = -2.0 // choice points = 45 // fails = 12
splitSL = 1

solver pass: 2
optimal solution? -> 0
time = 0.06800399999999929
objective value = -1.0 // choice points = 104 // fails = 70
splitSL = 1
solstatus == 0 -> none or no other solutions. do/while will break
Solver aktiv : Laufzeit=0000,1 s
thread gestoppt


print values:
splitSL = true
WOrt    = 0
WZeit   = 0 0 0
vo_zeit = 44 44 32 32 40 40 40 40 40 44 44 44
vo_tag  = 3 4 0 1 0 1 2 3 4 0 1 2
vo_hs   = 2 2 2 2 1 1 1 1 1 1 1 1

SoftConstraints:
[17-5, 22-5, 22-5]

Softconstraint 17 in Widerspruch mit Vorlesung 250083.2 [Analysis 1]
Beschreibung: Als Nachfolger definiertes Paar von Vorlesungen muss zeitlich unmittelbar anschließend stattfinden - es wird nicht derselbe Hörsaal erzwungen.

Softconstraint 22 in Widerspruch mit Vorlesung 250083.2 [Analysis 1]
Beschreibung: Die Pflichten (+) für einen bestimmten Hörsaal der Vorlesungen müssen eingehalten werden.

Softconstraint 22 in Widerspruch mit Vorlesung 250083.2 [Analysis 1]
Beschreibung: Die Pflichten (+) für einen bestimmten Hörsaal der Vorlesungen müssen eingehalten werden.


XML =
<?xml version="1.0" encoding="UTF-8"?>

<solution timestamp="31.01.2007 02:00:47">
  <obj value>-1.0</obj value>
  <constraints>660</constraints>
  <Variables>468</Variables>
  <WOrt>0</WOrt>
  <WZeit>0 0 0</WZeit>
  <vo_zeit>44 44 32 32 40 40 40 40 40 44 44 44</vo_zeit>
  <vo_tag>3 4 0 1 0 1 2 3 4 0 1 2</vo_tag>
  <vo_hs>2 2 2 2 1 1 1 1 1 1 1 1</vo_hs>
</solution>

keep connection alive thread abbrechen:

instanzen von solver und reporter beenden.
```

Abbildung 4.3: Screenshot – Echtzeitausgabe des Lösungsvorganges

0.Vorlesungsplan

Dokument automatisch erstellt: 21. Juni 2007 15:30
Zeit der Berechnung der Lösung: 21. Juni 2007 14:31

1.2.Hörsaal C206

1.2.1.Block A

```
Vorlesung 250082:1 Zeit = 08:00 Tag = Mon HSaal = C206
Vorlesung 250082:2 Zeit = 08:00 Tag = Die HSaal = C206
```

1.2.2.Block B

```
Vorlesung 250081:1 Zeit = 11:00 Tag = Don HSaal = C206
Vorlesung 250081:2 Zeit = 11:00 Tag = Fre HSaal = C206
Vorlesung 250082:1 Zeit = 08:00 Tag = Mon HSaal = C206
Vorlesung 250082:2 Zeit = 08:00 Tag = Die HSaal = C206
```

1.2.3.Block C

```
Vorlesung 250081:1 Zeit = 11:00 Tag = Don HSaal = C206
Vorlesung 250081:2 Zeit = 11:00 Tag = Fre HSaal = C206
Vorlesung 250082:1 Zeit = 08:00 Tag = Mon HSaal = C206
Vorlesung 250082:2 Zeit = 08:00 Tag = Die HSaal = C206
```

1.2.4.Block D

```
Vorlesung 250082:1 Zeit = 08:00 Tag = Mon HSaal = C206
Vorlesung 250082:2 Zeit = 08:00 Tag = Die HSaal = C206
```

2.2.prof2

Vorlesungsübersicht für prof2

2.2.1.Block A

```
Vorlesung 250082:1 Prof = prof2 Zeit = 08:00 Tag = Mon
HSaal = C206
Vorlesung 250082:2 Prof = prof2 Zeit = 08:00 Tag = Die
HSaal = C206
Vorlesung 250084:1 Prof = prof2 Zeit = 10:00 Tag = Fre
HSaal = HS1
Vorlesung 250084:2 Prof = prof2 Zeit = 11:00 Tag = Mon
HSaal = HS1
Vorlesung 250084:3 Prof = prof2 Zeit = 11:00 Tag = Die
HSaal = HS1
Vorlesung 250084:4 Prof = prof2 Zeit = 11:00 Tag = Mit
HSaal = HS1
```

Abbildung 4.4: Screenshot – PDF/HTML Ausgabe (auszugsweise)

5 Dokumentation des Programmes

5.1 Aufbau

Das Programm habe ich so aufgebaut, dass es so weit wie möglich aus klar voneinander getrennten Teilen besteht. Diese Module stehen jeweils für eine eigene Funktion und werden von den Webseiten aufgerufen. Die Servlets verarbeiten die eingegebenen Daten und stehen mit der zentralen Verwaltung der Daten in Verbindung. Die Aufbereitung und Ausgabe dynamischer Informationen wird zentral verwaltet. Es werden die Methoden, welche von außen aufgerufen werden, und die anderen Hilfsprozeduren in zwei Klassen getrennt, wobei die eine abstrakt ist und von der anderen erweitert wird. Außerdem gibt es eine Ebene für die Verwaltung der Daten und eine weitere stellt Hilfsfunktionen für die Webseiten zur Verfügung. Viele der Webseiten verwenden dynamische, sie begleitende Skripts, um zum Beispiel schon bei der Eingabe der Daten ungültige Werte abfangen zu können. Weiters muss darauf geachtet werden, dass jede Webseite nur nach erfolgter Authentifizierung ausgegeben werden darf und durch keine Eingabe in den Daten Inkonsistenzen erzeugt werden. Vor allem der letzte Punkt ist ein sehr wichtiges Thema, da es schwierig ist, alle erdenklichen Eingaben bei parallel laufenden Anfragen vorherzusehen.

5.1.1 Webseiten

Fast alle Webseiten bestehen aus dynamischen JSP („Java Server Pages“) Seiten. Das sind großteils normale HTML Seiten mit eingebettetem Code, der die dynamischen Elemente einbindet. Das Prinzip ist, dass alle diese Seiten zu JAVA Klassen umgewandelt werden, welche vom Webserver kompiliert und ausgeführt werden. Der eingebettete Code verhält sich dann nach der Umwandlung so, als ob er Teil eines normalen JAVA Programmes wäre. Viele Teile einer Webseite mit dynamischem Inhalt habe ich so gestaltet, dass eine externe Methode aufgerufen wird, welche den dafür notwendigen HTML Code auf Basis der gerade aktuellen Daten erzeugt. Freilich gibt es hierfür bessere Lösungen, jedoch hatte ich bei weitem nicht genug Erfahrung, um mit weiter entwickelten Werkzeugen umgehen zu können.

Die Webseiten selber sind in mehrere Abschnitte unterteilt. Das Wurzelverzeichnis beinhaltet allgemeine Informationen und eine Login-Maske. Nach einem erfolgreichen Login – das beinhaltet eine Überprüfung des Passworts – einen Test ob Cookies aktiviert sind und eine Überprüfung der JavaScript-Funktionalität ist je nach Authentifizierung nur ein beschränkter Teil erreichbar. Das ist eine Trennung in einen Teil für den Administrator, einen für die Vortragenden, einen für die Studenten und ein Test.

Mit Abstand die meisten Funktionen sind für den Administrator zugänglich. Er verwal-

tet fast alle Daten und hat die Kontrolle über die Lösungssuche durch den Solver und die Ansicht der Lösung. Studenten und Vortragende sind ähnlich, sie können Wünsche abgeben und Vortragende sind zusätzlich in der Lage, Daten über sich selbst anzugeben. Es ist somit alles auf eine zentrale Verwaltungsebene mit beschränktem Zugriff ausgelegt, und nur die notwendigsten Daten anderer werden extern gesammelt.

5.1.2 Struktur

Wie bereits einleitend erwähnt, gibt es einen modularen Aufbau. Das heißt, der Programmcode ist in kleine funktionale Einheiten geteilt, die wiederum zentral aufgerufen werden. Grob gesehen gibt es zwei Pakete. Das eine ist für zentrale Aufgaben, die grundlegenden Funktionen, die Verwaltung der Daten und Steuerung zuständig. Das andere Paket beinhaltet alle Servlets, welche mit dem Benutzer über die diversen Eingabemasken interagieren. Dabei rufen die Servlets Methoden aus den Klassen des anderen Pakets auf. Auch gibt es eine zentrale Anlaufstelle für alle Funktionen, die auf Übersichtsseiten für das Web aufbereitete Auszüge der Daten darstellen.

5.1.3 Sicherheit

Eine Sitzung läuft so ab, dass sich zuerst der Benutzer authentifizieren muss. Anschließend wird seiner Session eine Begleitvariable zugewiesen, die ihn von nun an nur in seinen für ihn bestimmten Bereich zulässt. Versucht er dies zu umgehen, wird das erkannt und automatisch umgeleitet. Ebenso verhält es sich mit Zugriffen durch nicht authentifizierte Benutzer.

Bei der Authentifizierung werden zwei Dinge überprüft: es müssen zur Speicherung der Benutzererkennung Cookies angelegt werden und zweitens muss JavaScript aktiviert sein, weil sonst manche Eingabemasken nicht vollständig angezeigt werden. Dies hat den Sinn, bei großen Masken, die viele identische Listen beinhalten, die Liste nur einmal zu übertragen, und diese wird mittels JavaScript dynamisch an jede notwendige Stelle im HTML Code kopiert. Außerdem werden, wenn möglich, clientseitig die eingegeben Daten auf Korrektheit überprüft. Diese Maßnahme reduziert serverseitig ebenfalls die aufkommenden Datenmengen.

5.1.4 Datenverwaltung

Die Verwaltung der Daten habe ich so gelöst, dass ich keine Datenbank verwende, sondern die gesamten Informationen in einer XML Datei abspeichere. Dies hat den Vorteil, dass sich mehrere Datensätze leicht verwalten lassen, es kann durch Kopieren einer einzigen Datei eine Sicherheitskopie erstellt werden und auch mittels eines Texteditors Änderungen vergenommen werden. Wegen des letzten Punktes habe ich darauf geachtet, eine klar verständliche Struktur zu wählen. Noch ausständig ist eine Validierung der gesamten Datenstruktur gegen ein XML Schema. Der Aufbau und die Struktur der XML Datei sieht grob gesagt so aus, dass es einen kleinen Teil für allgemeine Einstellungen gibt und dann drei größere Kategorien für Hörsäle, Vortragende und Studenten

– also Personen – und die Vorlesungen. In diesen Kategorien werden zum Beispiel die Zeitspannen für die zeitliche Verfügbarkeit vermerkt, das heißt, jeder Hörsaal und jeder Vortragende beinhaltet eine Liste von XML Elementen, welche für jeden Wochentag stehen und die Zeitspannen angeben. Genauso werden die Wünsche eines Vortragenden genau bei ihm eingetragen. Vorlesungen haben viel mehr assoziierte Daten. Daher gibt es eine feste Liste von Vorlesungen mit all ihren spezifischen Daten. Informationen über ihre Verknüpfungen, besondere sie charakterisierende Eigenschaften und Daten werden separat von ihnen gespeichert.

5.2 Entwicklungsablauf

5.2.1 Planung und Organisation

Da ich bei der Entwicklung auf keine andere Person angewiesen war, musste ich keinen Zeitplan einhalten und mich auch mit keinen anderen Projektmitarbeitern koordinieren.

Bei der Entwicklung der gesamten Anwendung ging ich in mehreren Schritten vor. Zuerst erarbeitete ich mir einen Überblick über den gesamten Funktionsumfang des `ILOG-Solvers`. Hierfür las ich das Handbuch und testete einige selbst gewählte Beispiele. Parallel entwickelte ich einen ersten Entwurf für das mathematische Modell, um die eigentliche Problemstellung erfassen zu können. Dabei zeigte sich, dass es mehrere Ansätze gibt und genauere Untersuchungen notwendig waren. Zuerst einmal galt es zu entscheiden, aus welcher Perspektive die zu modellierenden Daten betrachtet werden. Die natürlichste Art, zu der ich mich auch entschied, ist diejenige, bei welcher die Vorlesungen die Variablen sind. Es sind dann sowohl die Informationen zum konkreten Ort, Zeit und Tag als auch alle relevanten Daten für jede Vorlesung zu speichern. Da viele Vorlesungen an mehreren Tagen pro Woche stattfinden, musste ich einen Weg finden, dies ebenfalls modellieren zu können. Mein Weg war derjenige, alle Einheiten einer Vorlesung separat zu speichern. Das lag daran, dass zum Beispiel bei Proseminaren die Namen der Vortragenden für jede Einheit variieren.

Diese und viele andere Überlegungen führten letztendlich dazu, das mathematische Modell in einer ersten Version ausformulieren zu können. Der nächste Schritt war, die praktische Umsetzung mit Hilfe des `ILOG-Solver` durchzuführen. Nach vielen Iterationen des Ausprobierens, Testens und Anpassens veränderte sich auch die ursprüngliche Formulierung der Randbedingungen, und es gab einige neue Bedingungen, die ebenfalls erfasst werden mussten. Dazu zählen die Blocklehrveranstaltungen, die ich verallgemeinerte, damit sie sich zu allen anderen Vorlesungen homogen vertragen, und die genauere Ausarbeitung der Zielfunktion, die verwendet wird, um unterschiedlich gewichtete Wünsche erfassen zu können. Als endgültigen Test verwendete ich die realen Daten eines Semesters der mathematischen Fakultät der Universität Wien und definierte selbst gewählte zusätzliche Bedingungen und Wünsche. Anfangs war es unmöglich, eine Lösung zu erhalten, doch nach genaueren Tests der Funktionen des Solvers konnte ich zusätzlich eine Suchstrategie angeben, welche eigentlich eine starke Einschränkung an die Suchmethode des Solver ist. Aber nur so ließ sich ein Ergebnis ausgeben, und ich konnte mit dem

nächsten Schritt fortfahren.

Die Umsetzung als Webservice verlief ebenfalls in mehreren Schritten. Zuerst erweiterte ich meine bisherigen Java Kenntnisse innerhalb einer guten Woche um die, welche für das Erstellen eines Webservices notwendig sind. Danach führte ich ein paar Tests durch, um mit den technischen Details besser vertraut zu sein und erarbeitete parallel eine Methode, um über eine Schnittstelle mit dem `ILOG-Solver` kommunizieren zu können. Nachdem ich eine stark vereinfachte, nicht-dynamische Version der Verwaltung der Daten über das Webservice und die Schnittstelle fertig hatte, kombinierte ich beide Entwicklungszweige. Danach erweiterte ich dieses Kernprogramm schrittweise um dynamische Elemente. Diese Art der Entwicklung war für mich deshalb attraktiv, weil ich relativ schnell von einer funktionierenden Version auf eine nächste kam und durch diese Gliederung der Schritte schnell Fehler und Regressionen entdeckt werden können. Freilich war es später noch an manchen Stellen notwendig, Implementierungen zu verwerfen und sie durch andere zu ersetzen. Außerdem musste eine Eingabemaske umgeschrieben werden, damit sie mit größeren Datenmengen umgehen kann und auch andere nachträgliche Erweiterungen waren notwendig. Diese Arbeit an Details und die Fehlersuche stellte sich schnell als der Teil mit dem größten Arbeitsaufwand heraus. Nicht zu vergessen ist auch die parallel erfolgte Dokumentationsarbeit im und über das Programm.

5.2.2 Versionsmanagement

Im Lauf der Entwicklung des immer größer werdenden Programmes wurde es für mich wichtig, auf frühere funktionierende Versionen der Dateien Zugriff zu haben. Auch wollte ich parallel sowohl eine ältere funktionierende als auch eine aktuelle Version an der ich arbeiten konnte, zur Verfügung haben. Ein Versionsmanagementprogramm ist dabei eine Erweiterung des Dateisystems um eine weitere Dimension: die Zeit. Es ist möglich, frühere Versionen und vergangene Zustände aufzuzeichnen und darauf zuzugreifen. Anfänglich verwendete ich hierfür das Versionsmanagementprogramm `CVS`[1], weil ich damit schon einige Erfahrungen hatte. Ich musste jedoch bald feststellen, dass es bei meinen Anwendungen Probleme verursachte und sich damit nicht effizient arbeiten ließ. Das lag denke ich daran, dass es etwas älter ist und nur für einen speziellen Aufgabenbereich (Textdateien, kein Unicode) ausgelegt ist.

Später entschied ich mich für `Subversion`[2], da es für mich gegenüber `CVS` einige Vorteile hat. Einige der nützlichen Funktionen, die es zum Teil von `CVS` abhebt, sind die folgenden:

- Anstatt jede Datei einzeln zu versionieren, benützt es globale Revisionsnummern. Dadurch ist es einfacher, an einen speziellen Punkt in der Vergangenheit zurückzukehren. Auch ist die Sicherheit gegenüber Fehlern und Inkonsistenzen deutlich höher als bei `CVS`.
- Verallgemeinerung von „Tags“ und „Branches“. Das bedeutet, dass es keinen Unterschied zwischen einem neuen (Sub-)Entwicklungszweig und einer ausgezeichneten

[1] `http://www.nongnu.org/cvs/`
[2] `http://subversion.tigris.org/`

Versionsnummer gibt. Außerdem lassen sich problemlos für Unterverzeichnisse neue „Tags “ einführen, um so die Arbeit in LaTeX, das Programmieren mit dem `OPLStudio` und die Entwicklung des Webservices in Java unabhängig voneinander verwalten zu können, obwohl sich alles innerhalb derselben Versionskontrolle befindet.

- Verschieben und Kopieren von Dateien und Verzeichnissen ohne Verlust der Geschichte.
- Locks zum Sperren.
- Mischen von verschiedenen Entwicklerzweigen in den Unterverzeichnissen derselben lokalen Arbeitskopie.
- Management der Zeilenumbrüche. Windows und Linux/Unix Umgebungen verwenden unterschiedliche Kennzeichnungen für das Ende eine Zeile in einer Textdatei. `Subversion` kann damit umgehen und verwaltet sie.
- Attribute, auch binäre Daten, für jede verwaltete Datei in einer assoziativen Liste.
- Unterstützung für vollständige und inkrementelle Backups der Datenbank.
- Listen zum Ignorieren bestimmter Dateien und Verzeichnisse. Da sowohl beim Programmieren, als auch bei dem Schreiben mit LaTeX automatisch generierte Dateien entstehen, die nicht verwaltet werden müssen, können diese so aus dem Versionsmanagement ausgeklammert werden.

Ich verwendete es in der Art, dass ich nach dem Bearbeiten die Änderungen regelmäßig manuell einpflegte. Dabei erwies sich die Einbettung von `Subversion` in den Dateimanager `Nautilus` von `GNOME` [sof] durch die `Nautilus-Subversion-Skripte` [Scu] genauso praktisch, wie auch die Bedienung von der Kommandozeile aus ebenfalls keine Schwierigkeit darstellt.

5.2.3 Backupmanagement

Ein Fehler in der Festplatte führt dazu, dass plötzlich alle Daten verloren gehen. Dies lässt sich zwar nicht verhindern [PWB07], jedoch lassen sich Vorkehrungen treffen, damit ein Ausfall keinen Verlust der gesamten Arbeit darstellt. Ich verwendete hierfür zwei Backups, eines war ein „live“ Abbild der momentanen Arbeitsdateien, das andere ein kompletter `dump`, so wird ein vollständiger Speicherauszug genannt, des `Subversion` Repository. Das Subversion Repository selbst hinterlegte ich auf dem Server der Universität Wien, auf den ich mittels exportiertem ssh-Schlüssel leicht direkten Zugriff hatte. Das hatte überdies den Vorteil, dass Änderungen immer sofort auf den Server hochgeladen werden und schnell lokal beschädigte oder verlorengegangene Daten wiederhergestellt werden können. Das dabei verwendete Backupskript bereinigte überdies die Arbeitsumgebung, indem es temporäre Dateien löscht.

Hier das Skript, welches sich in der Wurzel des Arbeitsverzeichnisses befindet und sowohl das Programm als auch die Diplomarbeit an sich verwaltet:

Listing 5.1: Backup Shellskript / Linux

```
#!/bin/bash
# $Id: diplomarbeit_kap06_programm_v1.tex 226 2007-06-21 22:19:50Z schillh4 $
# author: Harald Schilly

login="schillh4@login.univie.ac.at"
bkbdir="~/svn/SVN_diplomarbeit/"

#delete old
find  diplomarbeit-*.bz2 -maxdepth 0 -exec rm -f {} \;

#remove temporary files
IFS=" "
DELPATTERN="diplomarbeit-schilly*.pdf *.ps *.bbl *.log *.lol *.aux *.dvi \
*.backup *~ *.war *.jar *.class"

for i in $DELPATTERN; do
   echo $i
   find . -iname $i -exec rm -f {} \;
done

#date
datum=`date +%Y%m%d-%H%M`

if [ -e LaTeX/diplomarbeit.pdf ]; then
   mv LaTeX/diplomarbeit.pdf diplomarbeit-schilly-$datum.pdf
fi

#create backups and compress
dname=diplomarbeit-$datum.bkb.tar
echo $dname
tar cf $dname --exclude 'TAG*' --exclude '.svn' ./
bzip2 -zv9 $dname

dname=diplomarbeit-SVNdump-$datum
echo $dname
# --deltas are binary differencies, may reduce filespace
ssh $login "svnadmin dump -q --deltas $bkbdir > $dname ; \
bzip2 -zv9 ~/svn/$dname"

#show some information
ls -lah diplomarbeit-*.bz2
```

6 Zusammenfassung und Ausblick

Diese Diplomarbeit zeigt, dass eine automatische Einteilung der Vorlesungen möglich ist. Obwohl dies anfangs noch nicht klar war, konnten die Tests zur Überprüfung meiner Annahmen und der Verfeinerung der Bedingungen dies zeigen.

Die in der Praxis gehandhabte Methode kann nur auf sehr wenige Randbedingungen eingehen. Der wahre Vorteil dieses Modells besteht darin, keine Bedingung zu übersehen und auf viele weitere Details, wie zum Beispiel die Überschneidungen, einzugehen.

Auch war es für mich relativ überraschend, dass dieses Modell problemlos auf mehrere Fakultäten oder die gesamte Universität ausgedehnt werden kann. Dies würde in der Praxis neue Möglichkeiten bei der Einteilung gemeinsam genutzter Hörsäle bieten.

Die Lösungsmethode kann noch in der Hinsicht verbessert werden, dass einige der Bedingungen nicht unbedingt erfüllt sein müssen. Es würde sich hier im Unterschied zu dem hier verwendeten Solver ein flexibleres Programm anbieten, welches die Anzahl der Verletzungen von ausgewählten Randbedingungen minimiert. Das hätte den Vorteil, dass es mehr mögliche Lösungen gibt.

7 Source-Code des Modells

7.1 zeitplanung.mod

Listing 7.1: Source-Code Zeitplanung Modell

```
/*********************************************************************************
OPL modell fuer vorlesungsstundenplanung
autor:     harald schilly
programm:  ilog solver, opl 3.7
version:   6 W B P G S
$Id: final_zeitplanung.modfile 206 2007-06-20 12:04:08Z schillh4 $

*********************************************************************************
*          COPYRIGHT (C) 2005-2006 by HARALD SCHILLY, Vienna, Austria           *
*                                                                               *
*                           ALL RIGHTS RESERVED                                 *
*                                                                               *
*       You are not allowed modify, distribute, copy, display, perform,exhibit, *
* transmit, reproduce, publish, license, create derivative works from, transfer *
* or  sell the content or any part thereof,  except as explicitly permitted and *
* otherwise noted by the open and written consent of Harald Schilly.            *
* You are not allowed  to remove any copyright,  trademark or other proprietary *
* notices from the content.                                                     *
*********************************************************************************/

/*********************************************************************************
 ENTWICKLUNGSGESCHICHTE

 HINTERGRUND
 Es gibt 8 unterschiedliche Entwicklerzweige, von denen Nummer 6 voll
 ausgearbeitet wurde.
 Nummer 7 und 8 beinhalten weiterfuehrende Ideen, die Aufgrund von Fehlern
 bei der Anbindung des Solvers durch JNI in JAVA nicht
 umgesetzt werden konnten.
 Ausserdem gibt es Ideen den Loesungsvorgang durch eine dynamische Grafik
 zu visualisieren.
 7 basiert auf Version 5, 8 auf Version 6.
 Nummer 1 bis 3 sind Testversionen in einer noch nicht verwendbaren
 Alphaversion.
 Nummer 4 und 5 sind separate Vorarbeiten fuer Version 6.

 Version 6 hat mehrere Unterversionen die durch Grossbuchstaben gekennzeichnet
 sind:
 Schluesselbegriffe:
   W = Wunschgruppen
   B = Blocklehrveranstaltungen
   P = Pflichten
   G = Hoersaal-Gruppen
   S = Softconstraints
```

```
45
46  ****************************************************************************
47  CHANGELOGS DER VERSIONEN
48
49 * dokumentieren und Code bereinigen
50
51 Version 6WBPGS basiert auf Version 6WBPG
52 * alles gleich, aber alle bestimmenden Gruppen von Constraints bekommen eine
53   oder-Verknuepfung mit einer binaeren Variable, welche in der Zielfunktion
54   minimiert wird -> simuliert sogenannte Softconstraints.
55
56 Version 6WBPG basiert auf version 6WBP
57 * dauer -> minuten, flexibler, weniger Variablen!!!
58 * Gruppen f Hoersaele
59 ** mit Zeiten fuer Ortswechsel? - pause[4] - verworfen - Ortswechselmatrix
60
61 Version 6WBP  basiert auf version 6WB
62 * mehrere Profs=metaProf und alle muessen bei der VO anwesen sein
63   und haben dann keine zeit fuer eine andere VO
64 * Bedingung fuer selben HS bei Nachfolger auf allgemeine
65   Kopplungsbedingung verschoben
66 * Ausschlusszeiten wie fuer Hoersaele auch fuer Vortragende
67 * spaeter fuer zeitvergleiche innerhalb woche
68 * diffDay fuer untersch. tage
69 ** Bedingungen zusammen: am nachfolgenden Tag
70 * Pflicht: Auslagerung der Wunschgruppe 1 auf separaten Constraint
71 * Normalisierungen der Datenstruktur:
72 ** TZSZ und TAAST eingefuehrt
73 ** Teile mit aufsteigender Option
74 * groesse der HS an maxAnzZH anpassen:
75   2 erzwungene Cluster: gross und klein mit automatischer Trennung
76
77 Version 6WB basiert auf version 6W
78 * zusaetzlich Verwaltung von Blocklehrveranstaltungen
79 * Wuensche fuer zeit/hs optional waehlbar
80
81 Version 6W basiert auf Version 6
82 * Wunschgruppen
83
84 Version 6  basiert auf version 4 und 5
85 * keine Liste von zeiten bei HS/Prof mehr, sondern dafuer
86   Matrix mit von bis pro Tag
87 * Erweiterung der prof-daten fuer Mittagspause
88 * Erweiterung um Wunschliste mit max-variablen/1 dim (spaeter verworfen)
89
90 Version 5 basiert auf Version 4
91 * erweiterung auf Uhrzeitformat <h:min> ... verworfen: siehe Version 6WBPG
92 * keine Liste von Zeiten bei HS/Prof mehr, sondern dafuer
93   Matrix mit von-bis pro Tag
94
95 Version 4
96 * Originaldaten werden eingelesen und umgerechnet.
97 * diesmal aber die zeiten pro Tag in 1/4 Stunden gerechnet.
98 * Neuformulierung der Urversion
99 ****************************************************************************/
100
101 /****************************************************************************
102        GLOBALE DATEN
```

```
103 ******************************************************************************/
104
105 // logische Basistypen
106 range Boolean [0..1];
107 range Trinary [-1..1];
108
109 // grundlegende Intervalle
110 range zeit_4 [0..23*4+3];
111 range rSoftC [0..30];
112
113 // grundlegende Daten aus .dat Datei einlesen
114 int+ pause[1..3]                                                    = ...;
115 enum Tage                                                           = ...;
116 enum TypVO                                                          = ...;
117 Boolean ueberschneid[0..card(TypVO)-1, 0..card(TypVO)-1]            = ...;
118 enum TypZeit                                                        = ...;
119 int+ modul                                                          = ...;
120 int+ splitSmallLarge                                               = ...;
121 {string} HSGruppen                                                 = ...;
122 int+ HSortswechsel[0..card(HSGruppen)-1,0..card(HSGruppen)-1]       = ...;
123 int+ SoftC_Thld_max                                                = ...;
124 rSoftC SoftC_Thld_cnt                                              = ...;
125
126 /******************************************************************************
127        DATENSTRUKTUREN
128 ******************************************************************************/
129
130 struct sZeitIntervall {
131    zeit_4 von;
132    zeit_4 bis;
133 };
134
135 struct sHSaal4 {
136    string            name;
137    int+              MaxAnzZuhoerer;
138    {string}          HSGruppe;
139    sZeitIntervall    zeit[Tage];
140 };
141
142 struct sVortragender4 {
143    string            name;
144    Boolean           mittagspause;
145    sZeitIntervall    zeit[Tage];
146 };
147
148 struct sVoID {
149    string name;
150    int+   teil;
151 };
152
153 struct sVorlesungB {
154    sVoID    ID;
155    string   profID;
156    int+     dauer;
157    string   titel;
158    {string} BGruppe;
159    int+     MaxAnzZH;
160    {string} HSGruppe;
```

```
161     TypVO    typVO;
162     TypZeit  typZeit;
163     Boolean  PCLAB;
164  };
165
166  struct sWunsch {
167     Boolean  bTag;
168     Tage     tag;
169     zeit_4   zeit;
170  };
171
172
173  struct sWunscheZeit {
174     sVoID       ID;
175     float       gewicht; // von -1 bis 1
176     {sWunsch}   wuensche;
177  };
178
179  struct sWunscheOrt {
180     sVoID       ID;
181     float       gewicht; // von -1 bis 1
182     {string}    hs;
183  };
184
185  struct sPflichten {
186     sVoID    ID;
187     Boolean  bTag;
188     Tage     tag;
189     zeit_4   zeit;
190     string   hs;
191  };
192
193  struct sSelbeZeit {
194     sVoID    ID1;
195     sVoID    ID2;
196     Trinary  tag;
197     Trinary  zeit;
198     Trinary  hs;
199  };
200
201  struct sHsBelegt {
202     string   hs;
203     Tage     tag;
204     zeit_4   von;
205     zeit_4   bis;
206  };
207
208  struct sProfBelegt {
209     string   name;
210     Tage     tag;
211     zeit_4   von;
212     zeit_4   bis;
213  };
214
215  struct sMetaProf {
216     string   name;
217     {string} profID;
218  };
```

```
219
220 struct sNachfolger {
221    sVoID     v1;
222    sVoID     v2;
223 };
224
225 struct sSpaeter {
226    sVoID     v1;
227    sVoID     v2;
228 };
229
230 struct sDiffDay {
231    sVoID    v1;
232    sVoID    v2;
233 };
234
235 struct result {
236    string   name;
237    int      teil;
238    string   titel;
239    string   profID;
240    Tage     tag;
241    float    zeit;
242    int      dauer;
243    string   hsaal;
244 };
245
246 struct softCstruct {
247    int+     constr;
248    int+     VoID;
249 };
250
251 /*****************************************************************************
252  Einlesen der weiteren Daten aus .dat Datei mit Datenstrukturen
253 *****************************************************************************/
254
255 {sHSaal4}         HSaal                    = ...;
256 {sVortragender4}  Prof                     = ...;
257 {sVorlesungB}     VOsetB                   = ...;
258 sZeitIntervall    TypZeitZeiten[TypZeit]   = ...;
259 sZeitIntervall    MP_zeit                  = ...;
260
261 {sWuenscheZeit} wuenscheZeit               = ...;
262 {sWuenscheOrt}  wuenscheOrt                = ...;
263
264 {sPflichten} pflichten                     = ...;
265 {sPflichten} ausschluss                    = ...;
266
267 {sSelbeZeit} selbeZeit                     = ...;
268
269 {sHsBelegt}   hsBelegt                     = ...;
270 {sProfBelegt} profBelegt                   = ...;
271
272 {sMetaProf}  metaProf                      = ...;
273
274 {sSpaeter}     spaeter                     = ...;
275 {sNachfolger}  nachfolger                  = ...;
276
```

```
277 {string}    TZSZ                          = ...;
278 {string}    TAAST                         = ...;
279 {string}    teileAufsteigend              = ...;
280
281 /*****************************************************************************
282  Daten verarbeiten: Arrays fuer weitere Verwendung konstruieren
283 *****************************************************************************/
284
285 range rVOB [1..card(VOsetB)];
286 sVorlesungB VOB [rVOB];
287 initialize forall(i in rVOB)
288    VOB[i] = VOsetB.next(VOsetB.first(),i-1);
289
290 range rHSaal [1..card(HSaal)];
291 sHSaal4 HS [rHSaal];
292 initialize forall(i in rHSaal)
293    HS[i] = HSaal.next(HSaal.first(),i-1);
294
295 range rProf [1..card(Prof)];
296 sVortragender4 PR [rProf];
297 initialize forall(i in rProf)
298    PR[i] = Prof.next(Prof.first(),i-1);
299
300 /*****************************************************************************
301  Datenkonsistenz und Sinnhaftigkeit ueberpruefen
302 *****************************************************************************/
303 // eindeutige IDs
304 assert
305    forall(ordered v1,v2 in VOsetB)
306       v1.ID <> v2.ID ;
307 assert
308    forall(ordered h1,h2 in HSaal)
309       h1.name <> h2.name;
310 assert
311    forall(ordered p1,p2 in Prof)
312       p1.name <> p2.name;
313
314 // Zeiten Hoersaelen: Ende spaeter als Beginn
315 assert
316   forall (h in HSaal)
317     forall (t in Tage)
318      ( h.zeit[t].von <= h.zeit[t].bis );
319
320 // Zeiten Vortragende: Ende spaeter als Beginn
321 assert
322   forall (p in Prof)
323     forall (t in Tage)
324      ( p.zeit[t].von <= p.zeit[t].bis );
325
326 // bei selben Identifikaitonsnamen, muss die Zeil-Zahl
327 // der aufsteigenden Vorlesungen aufsteigend sein
328 // fuer VOsetB
329 assert
330    forall (ordered v1, v2 in VOsetB: v1.ID.name = v2.ID.name)
331       (v1.ID.teil+1 <= v2.ID.teil);
332
333 // Vortragender muss exisiteren
334 assert
```

```
335    forall (v in VOsetB)
336       (sum (p in Prof) (p.name=v.profID) = 1 \/ v.profID.subString(0,4)="meta");
337
338 // Test fuer Wuensche
339 assert
340    forall(w in wuenscheZeit)
341       sum (v in VOsetB) (v.ID=w.ID) = 1;
342 assert
343    forall(w in wuenscheOrt)
344       sum (v in VOsetB) (v.ID=w.ID) = 1;
345 assert
346  forall(wSet in wuenscheOrt)
347    forall(h in wSet.hs)
348       sum (hs in HSaal) ((hs.name = h) & (h <> "none")) = 1;
349 assert
350    forall(w in wuenscheZeit)
351       (-1 <= w.gewicht <= 1) & (w.gewicht <> 0);
352 assert
353    forall(w in wuenscheOrt)
354       (-1 <= w.gewicht <= 1) & (w.gewicht <> 0);
355 assert
356    forall(wSet in wuenscheZeit)
357       forall(w in wSet.wuensche)
358          w.zeit mod modul = 0;
359 // Zeit und kein Tag nicht gleichzeitig, weil sonst automatisch immer erfuellt!
360 assert
361    forall(wSet in wuenscheZeit)
362       forall (w in wSet.wuensche)
363          (w.zeit <> 0) \/ (w.bTag <> 0);
364
365 // test fuer Pflichten ... wie wuenscheB
366 assert
367    forall(p in pflichten)
368       sum (v in VOsetB) (v.ID=p.ID) = 1;
369 assert
370    forall(p in pflichten : p.hs <> "none")
371       sum(h in HSaal) (h.name = p.hs) = 1;
372 assert
373    forall(p in pflichten)
374       p.zeit mod modul = 0;
375 // und ausschluss
376 assert
377    forall(p in ausschluss)
378       sum (v in VOsetB) (v.ID=p.ID) = 1;
379 assert
380    forall(p in ausschluss : p.hs <> "none")
381       sum(h in HSaal) (h.name=p.hs) =1;
382 assert
383    forall(p in ausschluss)
384       p.zeit mod modul = 0;
385 // test fuer nachfolger
386 assert
387    forall(n in nachfolger)
388       sum(v in VOsetB) (n.v1=v.ID) = 1;
389 assert
390    forall(n in nachfolger)
391       sum(v in VOsetB) (n.v2=v.ID) = 1;
392
```

```
393 // tests fuer selbeZeit
394 assert
395    forall(s in selbeZeit)
396       sum (v in VOsetB) (v.ID=s.ID1) = 1;
397 assert
398    forall(s in selbeZeit)
399       sum (v in VOsetB) (v.ID=s.ID2) = 1;
400
401 // Tests fuer die zeitliche Ausschlussliste der Hoersaele
402 // hs muss exisitieren
403 assert
404    forall(s in hsBelegt)
405       sum(h in HSaal) (h.name = s.hs) = 1;
406 // Zeit sinnvoll
407 assert
408    forall(s in hsBelegt)
409       s.von < s.bis;
410 // Ausschlusszeiten fuer prof
411 assert
412    forall(pb in profBelegt)
413       sum(p in Prof) (pb.name = p.name) = 1;
414 assert
415    forall(p in profBelegt)
416       p.von < p.bis;
417
418 // Ueberschneidungsmatrix: muss symmetrisch sein
419 assert
420   forall(ordered i,j in [0..card(TypVO)-1])
421     ueberschneid[i,j]=ueberschneid[j,i];
422
423 // Mittagspause Zeit
424 assert
425    MP_zeit.von < MP_zeit.bis;
426
427 // meta prof
428 // Identifikationsname beginnt immer mit meta
429 assert
430    forall(p in metaProf)
431       p.name.subString(0,4) = "meta";
432 // Eindeutigkeit
433 assert
434    forall(ordered p1,p2 in metaProf)
435       p1.name <> p2.name;
436 // Zusammengefasste Vortragende muessen existieren
437 assert
438    forall(e in metaProf)
439       forall(p in e.profID)
440          sum(prof in Prof) (p=prof.name) = 1;
441 assert
442    forall(v in VOsetB : v.profID.subString(0,4) = "meta")
443       sum(p in metaProf) (p.name=v.profID) = 1;
444
445 // Vorlesungsidentifikationen in spaeter muessen existieren
446 assert
447    forall(sp in spaeter)
448       sum(v1, v2 in VOsetB) (sp.v1 = v1.ID & sp.v2 = v2.ID) = 1;
449
450 // TZSZ
```

```
451 assert
452    forall(n in TZSZ)
453       sum(v in VOsetB) (n = v.ID.name & v.ID.teil = 1) = 1;
454
455 // TAAST
456 assert
457    forall(n in TAAST)
458       sum(v in VOsetB) (n = v.ID.name & v.ID.teil = 1) = 1;
459
460 // teileAufsteigend
461 assert
462    forall(n in teileAufsteigend)
463       sum(v in VOsetB) (n = v.ID.name & v.ID.teil = 1) = 1;
464
465 // HS Gruppen muss es geben
466 assert
467    forall(v in VOsetB)
468       forall(hg in v.HSGruppe)
469          sum(h in HSGruppen) (h = hg) = 1;
470
471 /*****************************************************************************
472  Variablen zur Loesungssuche
473 *****************************************************************************/
474
475 // simulierte Softconstraints fuer Wuensche+Zielfkt.
476 var Boolean WZeit[1..card(wunscheZeit)];
477 var Boolean WOrt [1..card(wunscheOrt)];
478
479 // simulierter Softconstraint fuer Trennung zwischen kleinen und grossen HSaelen
480 // Die Idee ist, auch Vorlesungen mit wenigen Hoerern ausnahmsweise in groszen
481 // Hoersaelen unterzubringen.
482 // Achtung: Loesungsstrategie setzt den Wert dieser Variablen fix auf 1
483 //          da ansonsten bei Tests keine Loesungen gefunden werden konnten.
484 var Boolean trennungSL;
485
486 // die eigentlich gesuchten Variablen
487 var zeit_4  vo_zeit[rVOB];
488 var Tage    vo_tag[rVOB];
489 var rHSaal  vo_hs[rVOB];
490
491 // binaere Softconstraint-Variablen fuer die Gruppen der Constraints
492 var    Boolean softC [rSoftC, rVOB];
493 range  rSoftC_Thld   [0..SoftC_Thld_max];
494 var    rSoftC_Thld   SoftC_Thld[rSoftC];
495
496 /*****************************************************************************
497  Hilfsvariablen zur Loesungssuche
498 *****************************************************************************/
499
500 // groeszter kleiner Hoersaal
501 int+ MaxSmallHS =
502   max(h in rHSaal : HS[h].MaxAnzZuhoerer <= splitSmallLarge) HS[h].MaxAnzZuhoerer;
503
504 // Listen der IDs
505 {string} HSaalIDs          = { HS[h].name | h in rHSaal };
506 {string} HSaalIDsSmall     = {HS[h].name | h in rHSaal :
507                               HS[h].MaxAnzZuhoerer <= MaxSmallHS };
508 {string} VOBIDs            = {VOB[v].ID.name | v in rVOB };
```

```
509 {string} VOBIDsSmall       = {VOB[v].ID.name | v in rVOB :
510                                VOB[v].MaxAnzZH <= MaxSmallHS };
511
512 // Ausgabe zur Information
513 display MaxSmallHS;
514 display card(VOsetB);
515 display card(HSaalIDs);
516 display card(HSaalIDsSmall);
517 display card(VOBIDs);
518 display card(VOBIDsSmall);
519
520 // anzTeile berechnen
521 {sVoID} VOBnameISname[v in rVOB]   = {VOB[z].ID | z in rVOB :
522                                       VOB[v].ID.name = VOB[z].ID.name};
523 int VOBanzTeile[v in rVOB]         = card(VOBnameISname[v]);
524
525 // AnzahlTafeln je Hoersaal muss ein Array fuer entsprechende Bedingung sein
526 int+ HSMaxAnzZuhoerer[i in rHSaal] = HS[i].MaxAnzZuhoerer;
527
528 // Blocklehrveranstaltungen: Matrix welche BlockVO gleichzeitig sind
529 {string} ueberschneidB[rVOB,rVOB];
530 initialize
531    forall(i,j in rVOB)
532       ueberschneidB[i,j] = VOB[i].BGruppe inter VOB[j].BGruppe;
533
534 // metaProf UEberschneidungsmatrix damit zwei VOs mit gleichen profs in metaProfs
535 // nicht zur selben Zeit stattfinden
536 range rMP [1..card(metaProf)];
537 sMetaProf MP [rMP];
538 initialize forall(i in rMP)
539    MP[i] = metaProf.next(metaProf.first(),i-1);
540 {string} MPueberschneid[rMP, rMP];
541 initialize
542    forall(i,j in rMP)
543       MPueberschneid[i,j] = MP[i].profID inter MP[i].profID;
544
545 range rHsBelegt [1..card(hsBelegt)];
546 sHsBelegt HsBelegt [rHsBelegt];
547 initialize forall(i in rHsBelegt)
548    HsBelegt[i] = hsBelegt.next(hsBelegt.first(),i-1);
549
550 // Wuensche fuer Ort und Zeit umrechnen in Arrays der Gewichte
551 // fuer Zuordnung auf Variable der Wunscherfuellung
552 range rWZeit [1..card(wenscheZeit)];
553 sWenscheZeit WlisteZeit [rWZeit];
554 initialize forall(i in rWZeit)
555    WlisteZeit[i] = wenscheZeit.next(wenscheZeit.first(),i-1);
556 range rWOrt [1..card(wenscheOrt)];
557 sWenscheOrt WlisteOrt [rWOrt];
558 initialize forall(i in rWOrt)
559    WlisteOrt[i] = wenscheOrt.next(wenscheOrt.first(),i-1);
560
561 // HS Gruppen ueberschneidung berechnen
562 {string} HSGselect[rVOB,rHSaal];
563 initialize
564    forall(i in rVOB & j in rHSaal)
565       HSGselect[i,j] = VOB[i].HSGruppe inter HS[j].HSGruppe;
566
```

```
567 /*****************************************************************************
568  EINSTELLUNGEN FUER SUCHALGORITHMUS
569 *****************************************************************************/
570
571 setting MIPEmphasis        = 1; // 4=hidden feas. 1=feas>opt.
572 setting MIPmethod_I        = 1;
573 setting aggCutLim          = 4; // default = 3
574 setting cutsFactor         = 2; // default = 4
575 setting barStartAl         = 4;
576 setting bbInterval         = 7;
577 setting bndStrenInd        = 1;
578 setting craInd             = 1;
579 setting nodeFileInd        = 3;
580 setting nodeSel            = 0;
581 setting perInd             = 1;
582 setting relaxPreInd        = 1;
583 setting aggInd             = -1;
584 setting aggFill            = 10; // default=10
585 setting depInd             = 3;
586 setting searchStrategy_I   = 0;  // 0=DF 1=SBS 2=DBDis 3=BFS 4=IDFS
587 setting symmetry           = 1;
588 setting preCompress        = 1;
589 //amount of probing on variables.
590 //reduces or increases solve time? 1=automatic 0 none 2,3,4 prov level 1 2 3
591 setting probe              = 1;
592 setting MIPOrdInd          = 1; // default = 1
593 //0=auto, -1 min, 1 maximum infeas.
594 //2 pseudo reduced costs, 3 Strong branching, 4 pseudo reduced costs
595 setting varSel             = 0;
596 setting strongCandLim      = 10;// default = 10 (list for strong branching)
597 setting strongThreadLim    = 2; // default = 1
598 setting diveType           = 0; // 0=auto
599 setting preslNd            = 1;
600 setting RINSHeur           = 20;
601 setting heurFreq           = 5;
602 setting cliques            = 0;
603 setting covers             = 0;
604 setting fracCuts           = 0;
605 setting disjCuts           = 0;
606 setting flowPaths          = 0;
607 setting flowCovers         = 0;
608 setting gubCovers          = 0;
609 setting MIRCuts            = 0;
610
611 setting workMem            = 256;
612
613 /*****************************************************************************
614  ZIELFUNKTION - gemeinsame Funktion fuer Softconstraints oder (exklusiv) Wuensche
615 *****************************************************************************/
616 // normalisieren
617 float WuenscheGewichtung =
618   sum(w in rWZeit) (abs(WlisteZeit[w].gewicht))
619 + sum(w in rWOrt ) (abs(WlisteOrt[w].gewicht));
620
621 maximize
622  (SoftC_Thld_max=0 \/ SoftC_Thld_cnt=0) *
623    ((WuenscheGewichtung/card(VOBIDsSmall))
624    * (  sum(v in rVOB : VOB[v].MaxAnzZH   <= MaxSmallHS)
```

```
625          (HSMaxAnzZuhoerer[vo_hs[v]]      <= MaxSmallHS)
626       - sum(v in rVOB : VOB[v].MaxAnzZH   <= MaxSmallHS)
627          (HSMaxAnzZuhoerer[vo_hs[v]]      >  MaxSmallHS)
628      )
629  + sum(w in rWZeit : WlisteZeit[w].gewicht > 0) (WlisteZeit[w].gewicht * WZeit[w])
630  + sum(w in rWOrt  : WlisteOrt[w].gewicht  > 0) (WlisteOrt[w].gewicht  * WOrt[w])
631  - sum(w in rWZeit : WlisteZeit[w].gewicht < 0) (WlisteZeit[w].gewicht * WZeit[w])
632  - sum(w in rWOrt  : WlisteOrt[w].gewicht  < 0) (WlisteOrt[w].gewicht  * WOrt[w]))
633  - sum(j in rSoftC) (SoftC_Thld[j])
634
635 /*****************************************************************************
636  Constraints - alle notwendigen Randbedingungen und Wuensche
637 *****************************************************************************/
638 subject to {
639 // Zeitauswahl einschraenken
640 // modul=2 bedeutet, dass in halbstuendlichen Schritten gesucht wird
641 forall(v in rVOB)
642    vo_zeit[v] mod modul = 0;
643
644 // Ueberlappung in selben Hoersaal vermeiden
645 // VOB <-> VOB ... auch >=1 bei VO<->VO und VO<->VOB
646 forall(ordered i, j in rVOB : card(ueberschneidB[i,j]) >= 1 )
647   ((vo_hs[i]=vo_hs[j] & vo_tag[i]=vo_tag[j]) =>
648      (vo_zeit[i]*15 + VOB[i].dauer + pause[1]
649       <= vo_zeit[j]*15
650    \/
651      vo_zeit[j]*15 + VOB[j].dauer + pause[1]
652       <= vo_zeit[i]*15)) \/ softC[1,i];
653
654 // belegte Hoersaele
655 forall (v in rVOB)
656  forall (h in rHSaal)
657    forall (b in rHsBelegt : HsBelegt[b].hs = HS[h].name)
658      ((vo_hs[v]=h & vo_tag[v]=HsBelegt[b].tag) =>
659          ((vo_zeit[v]*15 + VOB[v].dauer <= HsBelegt[b].von*15)
660      \/   (vo_zeit[v]                   >= HsBelegt[b].bis))) \/ softC[2,v];
661
662 // belegte Vortragende
663 // Fall 1: normale Vorlesung
664 forall (v in rVOB : VOB[v].profID.subString(0,4) <> "meta")
665   forall (bp in profBelegt : bp.name = VOB[v].profID)
666     ((vo_tag[v]=bp.tag) =>
667       ((vo_zeit[v]*15 + VOB[v].dauer <= bp.von*15)
668     \/ (vo_zeit[v]                       >= bp.bis))) \/ softC[3,v];
669
670 // Fall 2: meta prof in VO - muss fuer alle gelten
671 forall (v in rVOB : VOB[v].profID.subString(0,4) = "meta")
672  forall (mp in metaProf : mp.name = VOB[v].profID)
673    forall (p in mp.profID)
674      forall (bp in profBelegt : bp.name = p)
675        ((vo_tag[v]=bp.tag) =>
676         ((vo_zeit[v]*15 + VOB[v].dauer <= bp.von*15)
677      \/  (vo_zeit[v]                   >= bp.bis))) \/ softC[4,v];
678
679 // ein prof kann zur selben Zeit nur einen Vortrag halten
680 forall(ordered i, j in rVOB : VOB[i].profID = VOB[j].profID
681                & VOB[i].profID.subString(0,4) <> "meta"
682                & VOB[j].profID.subString(0,4) <> "meta"
```

```
683                & VOB[i].profID <> "none"
684                & VOB[j].profID <> "none"
685                & card(ueberschneidB[i,j]) >= 1 )
686    ((vo_tag[i]=vo_tag[j]) =>
687        (vo_zeit[i]*15 + VOB[i].dauer + pause[3]
688                  <= vo_zeit[j]*15
689        \/
690         vo_zeit[j]*15 + VOB[j].dauer + pause[3]
691                  <= vo_zeit[i]*15))             \/ softC[5,i];
692
693  // meta Professoren
694  // Fall 1: alle prof die angegeben wurden muessen anwensend sein
695  //        --> gleichzeitig = 1
696
697  // zeit der VO muss in allen zeitspannen der profs sein
698  forall(v in rVOB : VOB[v].profID.subString(0,4) = "meta")
699   forall(m in metaProf : VOB[v].profID = m.name) // <- nur der eine meta prof
700    forall(p in m.profID) // <-  alle meta prof
701     forall(ps in rProf : PR[ps].name = p) // <- umrechnen auf Index in PR
702      forall(t in Tage)
703        ((vo_tag[v]=t) =>
704         ( PR[ps].zeit[t].von <= vo_zeit[v]
705         & PR[ps].zeit[t].bis * 15 >=
706                           vo_zeit[v]*15 + VOB[v].dauer)) \/ softC[6,v];
707
708  // Vergleich von VOs mit meta prof <-> normal prof
709  forall(v1 in rVOB : VOB[v1].profID.subString(0,4) = "meta")
710   forall(m in metaProf : VOB[v1].profID = m.name)
711    forall(p in m.profID) // <- alle profs, die meta profs in v1 sind
712     forall(v2 in rVOB : VOB[v2].profID = p
713                       & card(ueberschneidB[v1,v2]) >= 1)
714        ((vo_tag[v1]=vo_tag[v2]) =>
715          (vo_zeit[v1]*15 + VOB[v1].dauer + pause[3]
716                <= vo_zeit[v2]*15
717         \/
718           vo_zeit[v2]*15 + VOB[v2].dauer + pause[3]
719                <= vo_zeit[v1]*15)) \/ softC[7,v1];
720
721  // sind bei zwei metaVOs mind. zwei prof in den meta prof gleich,
722  // so koennen sie nicht zur selben Zeit gehalten werden
723  forall(ordered v1,v2 in rVOB : VOB[v1].profID.subString(0,4) = "meta"
724                               & VOB[v2].profID.subString(0,4) = "meta"
725                               & card(ueberschneidB[v1,v2])   >= 1 )
726    forall(mp1,mp2 in rMP : MP[mp1].name=VOB[v1].profID
727                          & MP[mp2].name=VOB[v2].profID
728                          & card(MPueberschneid[mp1,mp2]) >= 1)
729       ((vo_tag[v1]=vo_tag[v2]) =>
730          (vo_zeit[v1]*15 + VOB[v1].dauer + pause[3]
731                <= vo_zeit[v2]*15
732         \/
733           vo_zeit[v2]*15 + VOB[v2].dauer + pause[3]
734                <= vo_zeit[v1]*15)) \/ softC[8,v1];
735
736
737
738  // Vorlesung innerhalb Hoersaal-Zeitspanne
739  forall(i in rVOB)
740    forall(h in rHSaal)
```

```
741     forall(t in Tage)
742   ((vo_hs[i]=h & vo_tag[i]=t) =>
743       ( HS[h].zeit[t].von <= vo_zeit[i]
744       & HS[h].zeit[t].bis * 15 >=
745                 vo_zeit[i]*15+VOB[i].dauer)) \/ softC[9,i];
746
747 // selbe Bedingung fuer VO<->prof
748 forall(i in rVOB)
749   forall(p in rProf : PR[p].name = VOB[i].profID
750                     & VOB[i].profID.subString(0,4) <> "meta")
751     forall(t in Tage)
752       ((vo_tag[i]=t) =>
753        ( PR[p].zeit[t].von <= vo_zeit[i]
754        & PR[p].zeit[t].bis * 15 >=
755                           vo_zeit[i] * 15 + VOB[i].dauer)) \/ softC[10,i];
756
757 // Mittagspause
758 forall(i in rVOB : VOB[i].profID.subString(0,4) <> "meta")
759    forall(p in rProf : PR[p].name = VOB[i].profID
760                      & PR[p].mittagspause=1) // <- genau 0 bis 1 mal
761      ((vo_zeit[i]                  >= MP_zeit.bis)
762    \/ (vo_zeit[i]*15+VOB[i].dauer <= MP_zeit.von*15)) \/ softC[11,i];
763
764 // Mittagspause bei meta prof
765 forall(i in rVOB : VOB[i].profID.subString(0,4) = "meta")
766  forall(mp in metaProf : mp.name = VOB[i].profID)
767   forall(ps in mp.profID)
768    forall(p in rProf : PR[p].name = ps
769                      & PR[p].mittagspause=1)
770      ((vo_zeit[i]                  >= MP_zeit.bis)
771    \/ (vo_zeit[i]*15+VOB[i].dauer <= MP_zeit.von*15)) \/ softC[12,i];
772
773 // Teile nicht am selben Tag. eventuell streng aufsteigend? - siehe naechste
774 forall(ordered i,j in rVOB : VOB[i].ID.name=VOB[j].ID.name
775                            & (TAAST.member(VOB[i].ID.name) = 0) )
776    (vo_tag[i] <> vo_tag[j]) \/ softC[13,i];
777
778 // Erweiterung: teileAufsteigend bei selber VO, wenn gewuenscht
779 forall (ta in teileAufsteigend)
780    forall (ordered v1, v2 in rVOB :  VOB[v1].ID.name = ta
781                                   &  VOB[v2].ID.name = ta
782                                   & (VOB[v1].ID.teil < VOB[v2].ID.teil) )
783       (vo_tag[v1] < vo_tag[v2]) \/ softC[14,v1];
784
785 // es muessen genuegend viele Sitzplaetze vorhanden sein
786 forall(i in rVOB)
787   (VOB[i].MaxAnzZH <= HSMaxAnzZuhoerer[vo_hs[i]]) \/ softC[15,i];
788
789 // Vorlesungen des selben Typs (Art) duerfen sich nicht ueberlappen,
790 // ausser sie sind zB Uebungen die untereinander ueberlappen aber nicht
791 // mit anderen Hauptvorlesungen
792 forall(ordered i,j in rVOB : ueberschneid[ord(VOB[i].typVO), ord(VOB[j].typVO)] = 1
793                            & card(ueberschneidB[i,j]) >= 1 )
794   ((vo_tag[i]=vo_tag[j]) =>
795     (vo_zeit[i]*15 + VOB[i].dauer + pause[1] <= vo_zeit[j]*15
796   \/ vo_zeit[j]*15 + VOB[j].dauer + pause[1] <= vo_zeit[i]*15)) \/ softC[16,i];
797
798 // einige Vorlesungen nur zu bestimmten Zeiten
```

```
799 // zB: jederzeit, morgens, vormittags, mittags, nachmittags, abends
800 forall(i in rVOB)
801    (vo_zeit[i]                 >= TypZeitZeiten[VOB[i].typZeit].von
802   & vo_zeit[i]*15+VOB[i].dauer <= TypZeitZeiten[VOB[i].typZeit].bis*15)
803                                                          \/ softC[17,i];
804
805 // nachfolger (unmittelbar)
806 forall (n in nachfolger)
807 forall (ordered v1,v2 in rVOB: n.v1 = VOB[v1].ID & VOB[v2].ID = n.v2)
808   ((vo_zeit[v1]*15 + VOB[v1].dauer + pause[2] + (60 mod modul) >= vo_zeit[v2]*15)
809  & vo_zeit[v1]<vo_zeit[v2]
810  & vo_tag[v1]=vo_tag[v2]) \/ softC[18,v1];
811    // Achtung: Auch selber Hoersaal braucht
812    // Extrabedingung, da zB VO und Labor-UE unterschiedlich sein koennen
813
814 // spaeter: v1 ist in der Woche zeitlich spaeter am selben Tag
815 // oder an einem der darauffolgenden
816 forall (sp in spaeter)
817    forall (v1, v2 in rVOB : VOB[v1].ID = sp.v1 & VOB[v2].ID = sp.v2)
818       (((vo_tag[v1] = vo_tag[v2]) =>
819         (vo_zeit[v1]*15 + VOB[v1].dauer <= vo_zeit[v2]))
820     &   vo_tag[v1] <= vo_tag[v2])    \/ softC[19,v1];
821
822 // PC Labor Bedingung
823 // fuer mehr Labors -> Auswahlvariable pro VO einfuehren:
824 // Wertbereich 0 (= kein Labor), 1..Anzahl LABs mit entspr. Implikationsbedingung
825 forall (ordered v1,v2 in rVOB : VOB[v1].PCLAB = 1 & VOB[v2].PCLAB = 1
826                               & card(ueberschneidB[v1,v2]) >= 1 )
827 ((vo_tag[v1] = vo_tag[v2]) =>
828     ((vo_zeit[v1]*15 + VOB[v1].dauer < vo_zeit[v2]*15)
829   \/ (vo_zeit[v2]*15 + VOB[v2].dauer < vo_zeit[v1]*15))) \/ softC[20,v1];
830
831 // TeileZSZ - Teile einer mehrteiligen VO sollen zur selben Zeit beginnen
832 forall (tzsz in TZSZ)
833 forall (ordered v1,v2 in rVOB : VOB[v1].ID.teil=1 // erste der Teile
834                               & VOB[v2].ID.teil>1 // damit verglichen
835                               & VOB[v1].ID.name = VOB[v2].ID.name // selbe VO
836                               & VOB[v1].ID.name = tzsz) // und wenn gewuenscht
837    (vo_zeit[v1]= vo_zeit[v2]  // dann haben diese Vorlesungen dieselbe Beginnzeit!
838   & vo_hs[v1]  = vo_hs[v2])   \/ softC[21,v1];   // und selben Hoersaal
839
840 // Wunscherfuellungen mit gewicht > 0
841 forall (w in rWZeit : WlisteZeit[w].gewicht > 0)
842  forall (v in rVOB : VOB[v].ID = WlisteZeit[w].ID)
843    sum(wt in WlisteZeit[w].wuensche) (
844         (((wt.zeit <> 0)    =>  (vo_zeit[v] = wt.zeit))
845        & ((wt.bTag =  1)    =>  (vo_tag[v]  = wt.tag)))
846                                      ) >= WZeit[w];
847
848 forall (w in rWOrt : WlisteOrt[w].gewicht > 0)
849  forall (v in rVOB : VOB[v].ID = WlisteOrt[w].ID)
850    sum(h in rHSaal : WlisteOrt[w].hs.member(HS[h].name))
851       (h = vo_hs[v]) = WOrt[w];
852
853 // gewicht < 0 -> Wunsche fuer Ausschluesse
854 // ist WZeit = 1, so muessen fuer Bedingung alle wahr sein:
855 // d.h. entweder Zeit oder Tag ungleich dem Gewuenschten
856 // ansonsten kann es auch trotzdem passen (>= !)
```

```
857 forall (w in rWZeit : WlisteZeit[w].gewicht < 0)
858  forall (v in rVOB : VOB[v].ID = WlisteZeit[w].ID)
859    sum(wt in WlisteZeit[w].wuensche) (
860         (((wt.zeit <> 0)    =>  (vo_zeit[v] <> wt.zeit))
861        & ((wt.bTag =  1)    =>  (vo_tag[v]  <> wt.tag)))
862                             ) >= WZeit[w]*card(WlisteZeit[w].wuensche);
863
864 forall (w in rWOrt : WlisteOrt[w].gewicht < 0)
865  forall (v in rVOB : VOB[v].ID = WlisteOrt[w].ID)
866    sum(h in rHSaal : WlisteOrt[w].hs.member(HS[h].name))
867       (h = vo_hs[v]) <> WOrt[w];
868
869 // Zwangsbedingung: Wuensche -> Pflichten
870 forall (p in pflichten)
871    forall(v in rVOB : VOB[v].ID = p.ID)
872       (((p.zeit <> 0) => (vo_zeit[v] = p.zeit))
873      & ((p.bTag =  1) => (vo_tag[v]  = p.tag))) \/ softC[22,v];
874
875 forall (p in pflichten)
876    forall (v in rVOB : VOB[v].ID = p.ID)
877       forall (h in rHSaal : HS[h].name = p.hs)
878          ((p.hs <> "none") => (vo_hs[v] = h)) \/ softC[23,v];
879
880 // Pflichten umgekehrt -> Ausschluesse
881 forall (p in ausschluss)
882    forall(v in rVOB : VOB[v].ID = p.ID)
883      (((p.zeit <> 0) => (vo_zeit[v] <> p.zeit))
884     & ((p.bTag =  1) => (vo_tag[v]  <> p.tag))) \/ softC[24,v];
885
886 forall (p in ausschluss)
887  forall (v in rVOB : VOB[v].ID = p.ID)
888   forall (h in rHSaal : HS[h].name = p.hs)
889       ((p.hs <> "none") => (vo_hs[v] <> h)) \/ softC[25,v];
890
891 // Test: kleinere VO nicht in grossen HS
892 forall (v in rVOB : VOB[v].MaxAnzZH <= MaxSmallHS)
893    ((trennungSL = 1) => (HSMaxAnzZuhoerer[vo_hs[v]] <= MaxSmallHS)) \/ softC[26,v];
894
895 // selbeZeit - Kopplungsbedingung
896 forall (sz in selbeZeit)
897    forall (v1,v2 in rVOB : VOB[v1].ID = sz.ID1 & VOB[v2].ID = sz.ID2)
898       ((((sz.tag=1)   => (vo_tag[v1]  =  vo_tag[v2]))
899      & ((sz.zeit=1)  => (vo_zeit[v1] =  vo_zeit[v2]))
900      & ((sz.hs=1)    => (vo_hs[v1]   =  vo_hs[v2]))
901      & ((sz.tag=-1)  => (vo_tag[v1]  <> vo_tag[v2]))
902      & ((sz.zeit=-1) => // keine Ueberschneidung!
903           ( vo_tag[v1] = vo_tag[v2] =>
904                ( vo_zeit[v1]*15 + VOB[v1].dauer <= vo_zeit[v2]*15
905        \/       vo_zeit[v2]*15 + VOB[v2].dauer <= vo_zeit[v1]*15)))
906      & ((sz.hs=-1)   => (vo_hs[v1]   <> vo_hs[v2])))) \/ softC[27,v1];
907
908 // Hoersaal nur aus Vereinigung der thematischen Mengen waehlen
909 forall(v in rVOB & h in rHSaal : card(HSGselect[v,h]) = 0)
910    (vo_hs[v] <> h) \/ softC[28,v];
911
912 // Dauer fuer Standortwechsel
913 forall(v1,v2 in rVOB : card(ueberschneidB[v1,v2]) >= 1
914                     & VOB[v1].profID = VOB[v2].profID)
```

```
915  ((vo_tag[v1] = vo_tag[v2]) =>
916   (  (vo_zeit[v1] + VOB[v1].dauer - vo_zeit[v2]
917         >= min(g1 in VOB[v1].HSGruppe & g2 in VOB[v2].HSGruppe)
918               (HSortswechsel[ord(HSGruppen,g1),ord(HSGruppen,g2)]))
919  \/  (vo_zeit[v2] + VOB[v2].dauer - vo_zeit[v1]
920         >= min(g1 in VOB[v1].HSGruppe & g2 in VOB[v2].HSGruppe)
921               (HSortswechsel[ord(HSGruppen,g1),ord(HSGruppen,g2)]))))
922                                                    \/ softC[29,v1];
923
924 // Standortwechsel metaprof
925 forall(v1,v2 in rVOB : card(ueberschneidB[v1,v2]) >= 1
926                      & VOB[v1].profID.subString(0,4)="meta"
927                      & VOB[v2].profID.subString(0,4)="meta")
928  forall(mp1,mp2 in rMP :  MP[mp1].name=VOB[v1].profID
929                         & MP[mp2].name=VOB[v2].profID
930                         & card(MPueberschneid[mp1,mp2]) >= 1)
931  ((vo_tag[v1] = vo_tag[v2]) =>
932   (  (vo_zeit[v1] + VOB[v1].dauer - vo_zeit[v2]
933         >= min(g1 in VOB[v1].HSGruppe & g2 in VOB[v2].HSGruppe)
934               (HSortswechsel[ord(HSGruppen,g1),ord(HSGruppen,g2)]))
935  \/  (vo_zeit[v2] + VOB[v2].dauer - vo_zeit[v1]
936         >= min(g1 in VOB[v1].HSGruppe & g2 in VOB[v2].HSGruppe)
937               (HSortswechsel[ord(HSGruppen,g1),ord(HSGruppen,g2)]))))
938                                                    \/ softC[30,v1];
939
940 // Softconstraints
941 sum(j in rSoftC) SoftC_Thld[j]   <= SoftC_Thld_cnt;
942 forall(j in rSoftC)
943    sum(i in rVOB) softC[j,i]     <= SoftC_Thld[j];
944
945 }; // Ende der Constraints
946
947 /*****************************************************************************
948  SUCHSTRATEGIE
949 *****************************************************************************/
950 // das hier ist eine stark vereinfachte Version, gibt viele Variationen
951 // und andere erzwungene Massnahmen. Jedoch habe ich keine wirklich
952 // zufriedenstellende Strategie gefunden. Informationen aus dem Handbuch
953 // sind viel zu unergiebig.
954 // Diese Strategie sortiert teilweise dynamisch die Reihenfolge
955 // der Zuweisungen an die Variablen, wobei Zeit, Tag und Hoersaal der
956 // betrachteten Vorlesung gleichzeitig gewaehlt wird.
957
958 search {
959  trennungSL = 1;
960
961  forall(v in rVOB ordered by decreasing <
962                VOBanzTeile[v],
963                TZSZ.member(VOB[v].ID.name),
964                TAAST.member(VOB[v].ID.name),
965                VOB[v].dauer,
966                VOB[v].MaxAnzZH,
967                -dsize(vo_zeit[v])-dsize(vo_tag[v])-dsize(vo_hs[v])>) {
968    if not bound(vo_zeit[v]) then generate(vo_zeit[v]) endif;
969    if not bound(vo_tag[v])  then generate(vo_tag[v])  endif;
970    if not bound(vo_hs[v])   then generate(vo_hs[v])   endif;
971
972  };// end forall(rVOB)
```

```
973 }; // end search
974
975 /*****************************************************************************
976  AUSGABE und DATENAUFBEREITUNG
977 *****************************************************************************/
978
979 {result} res = { <VOB[i].ID.name , VOB[i].ID.teil , VOB[i].titel, VOB[i].profID,
980     vo_tag[i] , vo_zeit[i]/4.0 , VOB[i].dauer , HS[vo_hs[i]].name > | i in rVOB };
981
982 {softCstruct} softCset = { < i, j > | i in rSoftC, j in rVOB : ((softC[i,j]=1))};
983
984 display res;
985 display WZeit;
986 display WOrt;
987 display trennungSL;
988 display SoftC_Thld;
989 display softCset;
```

7.2 zeitplanung.dat

Listing 7.2: Source-Code Zeitplanung Daten (Beispiel)

```
SoftC_Thld_max = 0;
SoftC_Thld_cnt = 0;
Tage = {Mon Die Mit Don Fre};
modul = 2;
splitSmallLarge = 50;
pause = [ 0 0 0 ];
TypZeit = {
jederzeit tagsueber morgens vormittags mittags nachmittags spaetertag abends
};
TypZeitZeiten = [
< 32 84 > < 36 76 > < 32 48 > < 40 50 > < 44 52 > < 48 72 > < 56 72 > < 68 82 >
];
HSGruppen = { "UZA2" "UZA4" "UZA4o" "STDLHG" };
HSortswechsel = [
[ 0 5 10 20 ]
[ 5 0 5 20 ]
[ 5 5 0 20 ]
[ 25 25 30 0 ]
];
HSaal = {
< "HS1" 150 { "UZA2" } [ < 32 80 > < 32 70 > < 32 74 > < 32 84 > < 36 74 > ] >
< "C206" 35 { "UZA4" } [ < 0 94 > < 0 94 > < 0 94 > < 0 94 > < 0 94 > ] >
< "HS3" 100 { "UZA2" } [ < 0 94 > < 0 94 > < 0 94 > < 0 94 > < 0 94 > ] >
< "C101" 25 { "UZA4" } [ < 0 94 > < 0 94 > < 0 94 > < 0 94 > < 0 94 > ] >
};
MP_zeit = < 48 52 >;
Prof = {
< "prof1" 0 [ < 0 95 > < 0 95 > < 0 95 > < 0 95 > < 0 95 > ] >
< "prof2" 0 [ < 0 95 > < 0 95 > < 0 95 > < 0 95 > < 0 95 > ] >
< "prof3" 0 [ < 0 95 > < 0 95 > < 0 95 > < 0 95 > < 0 95 > ] >
< "prof4" 0 [ < 0 95 > < 0 95 > < 0 95 > < 0 95 > < 0 95 > ] >
};
TypVO = { frei AnfAllg Anf1 Anf2 PSA1 PSA2
 FortAllg FortSEAnalysis FortAlgebra FortAnalysis FortNumOpt FortBio
 FortStatistik FortDiffGeo FortKombi
 LAK PSLAK LAK2 Abend Ph PSPh };
ueberschneid = [
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]
[ 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ]
[ 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ]
[ 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]
[ 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ]
[ 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ]
[ 0 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 ]
[ 0 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 ]
[ 0 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 ]
[ 0 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 ]
[ 0 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 ]
[ 0 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 ]
[ 0 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 ]
[ 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 ]
[ 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ]
[ 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 ]
[ 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ]
```

```
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 ]
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 ]
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 ]
[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 ]
];
VOsetB = {
< < "250081" 1 > "prof1" 90 "PS Analysis 1" { "C" "B" } 20
  { "UZA4" "UZA2" } PSA1 tagsueber 0>
< < "250081" 2 > "prof2" 90 "PS Analysis 1" { "C" "B" } 20
  { "UZA4" "UZA2" } PSA1 tagsueber 0>
< < "250082" 1 > "prof2" 90 "PS LinAlg 1" { "D" "C" "B" "A" } 30
  { "UZA4" "UZA2" } PSA1 jederzeit 0>
< < "250082" 2 > "prof2" 90 "PS LinAlg 1" { "D" "C" "B" "A" } 30
  { "UZA4" "UZA2" } PSA1 jederzeit 0>
< < "250083" 1 > "prof1" 45 "Analysis 1" { "D" "C" "B" "A" } 70
  { "UZA2" } Anf1 vormittags 0>
< < "250083" 2 > "prof1" 45 "Analysis 1" { "D" "C" "B" "A" } 70
  { "UZA2" } Anf1 vormittags 0>
< < "250083" 3 > "prof1" 45 "Analysis 1" { "D" "C" "B" "A" } 70
  { "UZA2" } Anf1 vormittags 0>
< < "250083" 4 > "prof1" 45 "Analysis 1" { "D" "C" "B" "A" } 70
  { "UZA2" } Anf1 vormittags 0>
< < "250084" 1 > "prof2" 45 "LinAlg 1" { "D" "C" "B" "A" } 70
  { "UZA2" } Anf1 vormittags 0>
< < "250084" 2 > "prof2" 45 "LinAlg 1" { "D" "C" "B" "A" } 70
  { "UZA2" } Anf1 vormittags 0>
< < "250084" 3 > "prof2" 45 "LinAlg 1" { "D" "C" "B" "A" } 70
  { "UZA2" } Anf1 vormittags 0>
< < "250084" 4 > "prof2" 45 "LinAlg 1" { "D" "C" "B" "A" } 70
  { "UZA2" } Anf1 vormittags 0>
};
wuenscheZeit = {
< < "250081" 1 > 0.01 { < 1 Mon 0 > } > };
wuenscheOrt = {
< < "250081" 1 > 0.01 { "HS1" } > };
pflichten = {
< < "250083" 1 > 1 Mon 36 "none" >
< < "250083" 1 > 1 Mon 32 "none" >
};
ausschluss = {
< < "250084" 1 > 0 Mon 0 "C206" >
};
TAAST = { "250081" };
TZSZ = {};
teileAufsteigend = { "250081" };
selbeZeit = {
< < "250081" 1 > < "250081" 2 > -1 0 0 >
< < "250081" 1 > < "250081" 2 > -1 0 0 >
};
nachfolger = {
< < "250084" 1 > < "250082" 1 > >
};
spaeter = {
< < "250082" 1 > < "250082" 2 > >
};
hsBelegt = {
< "C206" Die 52 66 >
};
```

```
113 profBelegt = {
114 < "prof1" Die 52 72 >
115 };
116 metaProf = {
117 < "metagrp1" { "prof3" "prof1" } >
118 };
```

7.3 zeitplanung.xml

Das ist ein Beispiel für die XML-Rohdaten, welche dem Webinterface zu Grunde liegen und aus denen die `.dat`-Datei für den Solver generiert wird.

Listing 7.3: Source-Code XML Daten (Beispiel)

```
<?xml version="1.0" encoding="UTF-8"?>
<data version="1.0">
  <TimeModul value="2"/>
  <SplitSmallLarge value="50"/>
  <Tage value="Mon Die Mit Don Fre"/>
  <Bloecke value="A B C D"/>
  <ZeitTypen>
    <ZeitTyp zeittypID="jederzeit"   von="32" bis="84"/>
    <ZeitTyp zeittypID="tagsueber"   von="36" bis="76"/>
    <ZeitTyp zeittypID="morgens"     von="32" bis="48"/>
    <ZeitTyp zeittypID="vormittags"  von="40" bis="50"/>
    <ZeitTyp zeittypID="mittags"     von="44" bis="52"/>
    <ZeitTyp zeittypID="nachmittags" von="48" bis="72"/>
    <ZeitTyp zeittypID="spaetertag"  von="56" bis="72"/>
    <ZeitTyp zeittypID="abends"      von="68" bis="82"/>
  </ZeitTypen>
  <HSortswechsel>
    <HSGruppen value="UZA2 UZA4 UZA4o STDLHG"/>
    <matrix>
      <row value="0 5 10 20"/>
      <row value="5 0 5 20"/>
      <row value="5 5 0 20"/>
      <row value="25 25 30 0"/>
    </matrix>
  </HSortswechsel>
  <Raeume>
    <HS hsID="HS1" MaxAnzZH="150" RaumGruppen="UZA2">
      <Zeiten>
        <Tag tagID="Mon" von="32" bis="80"/>
        <Tag tagID="Die" von="32" bis="70"/>
        <Tag tagID="Mit" von="32" bis="74"/>
        <Tag tagID="Don" von="32" bis="84"/>
        <Tag tagID="Fre" von="36" bis="74"/>
      </Zeiten>
    </HS>
    <HS hsID="C206" MaxAnzZH="35" RaumGruppen="UZA4">
      <Zeiten>
        <Tag tagID="Mon" von="0" bis="94"/>
        <Tag tagID="Die" von="0" bis="94"/>
        <Tag tagID="Mit" von="0" bis="94"/>
        <Tag tagID="Don" von="0" bis="94"/>
        <Tag tagID="Fre" von="0" bis="94"/>
      </Zeiten>
      <Belegt>
        <Tag tagID="Die" von="52" bis="66"/>
      </Belegt>
    </HS>
    <HS hsID="HS3" MaxAnzZH="100" RaumGruppen="UZA2">
      <Zeiten>
        <Tag tagID="Mon" von="0" bis="94"/>
        <Tag tagID="Die" von="0" bis="94"/>
```

```
52          <Tag tagID="Mit" von="0" bis="94"/>
53          <Tag tagID="Don" von="0" bis="94"/>
54          <Tag tagID="Fre" von="0" bis="94"/>
55        </Zeiten>
56      </HS>
57      <HS hsID="C101" MaxAnzZH="25" RaumGruppen="UZA4">
58        <Zeiten>
59          <Tag tagID="Mon" von="0" bis="94"/>
60          <Tag tagID="Die" von="0" bis="94"/>
61          <Tag tagID="Mit" von="0" bis="94"/>
62          <Tag tagID="Don" von="0" bis="94"/>
63          <Tag tagID="Fre" von="0" bis="94"/>
64        </Zeiten>
65      </HS>
66    </Raeume>
67    <Vortragende>
68      <Mittagspause von="48" bis="52"/>
69      <Meta>
70        <MetaProf metaID="grp1" profs="prof3 prof1"/>
71      </Meta>
72      <Prof profID="prof1" pause="0" gewicht="0.4">
73        <Zeiten>
74          <Tag tagID="Mon" von="0" bis="95"/>
75          <Tag tagID="Die" von="0" bis="95"/>
76          <Tag tagID="Mit" von="0" bis="95"/>
77          <Tag tagID="Don" von="0" bis="95"/>
78          <Tag tagID="Fre" von="0" bis="95"/>
79        </Zeiten>
80        <Belegt>
81          <Tag tagID="Die" von="52" bis="72"/>
82        </Belegt>
83        <Wuensche/>
84      </Prof>
85      <Prof profID="prof2" pause="0" gewicht="0.6">
86        <Zeiten>
87          <Tag tagID="Mon" von="0" bis="95"/>
88          <Tag tagID="Die" von="0" bis="95"/>
89          <Tag tagID="Mit" von="0" bis="95"/>
90          <Tag tagID="Don" von="0" bis="95"/>
91          <Tag tagID="Fre" von="0" bis="95"/>
92        </Zeiten>
93        <Belegt/>
94        <Wuensche/>
95      </Prof>
96      <Prof profID="prof3" pause="0" gewicht="0.8">
97        <Zeiten>
98          <Tag tagID="Mon" von="0" bis="95"/>
99          <Tag tagID="Die" von="0" bis="95"/>
100         <Tag tagID="Mit" von="0" bis="95"/>
101         <Tag tagID="Don" von="0" bis="95"/>
102         <Tag tagID="Fre" von="0" bis="95"/>
103       </Zeiten>
104       <Belegt/>
105       <Wuensche/>
106     </Prof>
107     <Prof profID="prof4" pause="0" gewicht="0.2">
108       <Zeiten>
109         <Tag tagID="Mon" von="0" bis="95"/>
```

```
110         <Tag tagID="Die" von="0" bis="95"/>
111         <Tag tagID="Mit" von="0" bis="95"/>
112         <Tag tagID="Don" von="0" bis="95"/>
113         <Tag tagID="Fre" von="0" bis="95"/>
114       </Zeiten>
115       <Belegt/>
116       <Wuensche/>
117     </Prof>
118   </Vortragende>
119   <Studenten gewicht="0.1"/>
120   <Vorlesungen>
121     <TypVo>frei AnfAllg Anf1 Anf2 PSA1 PSA2
122           FortAllg FortSEAnalysis FortAlgebra FortAnalysis FortNumOpt
123           FortBio FortStatistik FortDiffGeo FortKombi
124           LAK PSLAK LAK2 Abend Ph PSPh
125     </TypVo>
126     <Ueberschneidungen>
127       <matrix>
128         <row value="0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0"/>
129         <row value="0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0"/>
130         <row value="0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0"/>
131         <row value="0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0"/>
132         <row value="0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0"/>
133         <row value="0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0"/>
134         <row value="0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0"/>
135         <row value="0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0"/>
136         <row value="0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0"/>
137         <row value="0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0"/>
138         <row value="0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0"/>
139         <row value="0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0"/>
140         <row value="0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0"/>
141         <row value="0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0"/>
142         <row value="0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0"/>
143         <row value="0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0"/>
144         <row value="0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0"/>
145         <row value="0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0"/>
146         <row value="0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0"/>
147         <row value="0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1"/>
148         <row value="0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0"/>
149       </matrix>
150     </Ueberschneidungen>
151     <VOs>
152       <VO VoID="250081" teil="1" profID="prof1" dauer="90" titel="PS Analysis 1"
153        Bloecke="C B" Ortswahl="UZA4 UZA2" MaxAnzZH="20" TypVO="PSA1"
154        TypZeit="tagsueber" PCLAB="0"/>
155       <VO VoID="250081" teil="2" profID="prof2" dauer="90" titel="PS Analysis 1"
156        Bloecke="C B" Ortswahl="UZA4 UZA2" MaxAnzZH="20" TypVO="PSA1"
157        TypZeit="tagsueber" PCLAB="0"/>
158       <VO VoID="250082" teil="1" profID="prof2" dauer="90" titel="PS LinAlg 1"
159        Bloecke="D C B A" Ortswahl="UZA4 UZA2" MaxAnzZH="30" TypVO="PSA1"
160        TypZeit="jederzeit" PCLAB="0"/>
161       <VO VoID="250082" teil="2" profID="prof2" dauer="90" titel="PS LinAlg 1"
162        Bloecke="D C B A" Ortswahl="UZA4 UZA2" MaxAnzZH="30" TypVO="PSA1"
163        TypZeit="jederzeit" PCLAB="0"/>
164       <VO VoID="250083" teil="1" profID="prof1" dauer="45" titel="Analysis 1"
165        Bloecke="D C B A" Ortswahl="UZA2" MaxAnzZH="70" TypVO="Anf1"
166        TypZeit="vormittags" PCLAB="0"/>
167       <VO VoID="250083" teil="2" profID="prof1" dauer="45" titel="Analysis 1"
```

```
168         Bloecke="D C B A" Ortswahl="UZA2" MaxAnzZH="70" TypVO="Anf1"
169         TypZeit="vormittags" PCLAB="0"/>
170       <VO VoID="250083" teil="3" profID="prof1" dauer="45" titel="Analysis 1"
171         Bloecke="D C B A" Ortswahl="UZA2" MaxAnzZH="70" TypVO="Anf1"
172         TypZeit="vormittags" PCLAB="0"/>
173       <VO VoID="250083" teil="4" profID="prof1" dauer="45" titel="Analysis 1"
174         Bloecke="D C B A" Ortswahl="UZA2" MaxAnzZH="70" TypVO="Anf1"
175         TypZeit="vormittags" PCLAB="0"/>
176       <VO VoID="250084" teil="1" profID="prof2" dauer="45" titel="LinAlg 1"
177         Bloecke="D C B A" Ortswahl="UZA2" MaxAnzZH="70" TypVO="Anf1``
178         TypZeit="vormittags" PCLAB="0"/>
179       <VO VoID="250084" teil="2" profID="prof2" dauer="45" titel="LinAlg 1"
180         Bloecke="D C B A" Ortswahl="UZA2" MaxAnzZH="70" TypVO="Anf1"
181         TypZeit="vormittags" PCLAB="0"/>
182       <VO VoID="250084" teil="3" profID="prof2" dauer="45" titel="LinAlg 1"
183         Bloecke="D C B A" Ortswahl="UZA2" MaxAnzZH="70" TypVO="Anf1"
184         TypZeit="vormittags" PCLAB="0"/>
185       <VO VoID="250084" teil="4" profID="prof2" dauer="45" titel="LinAlg 1"
186         Bloecke="D C B A" Ortswahl="UZA2" MaxAnzZH="70" TypVO="Anf1"
187         TypZeit="vormittags" PCLAB="0"/>
188     </VOs>
189     <Pflichten>
190       <Pflicht VoID="250083" teil="1" parity="+" tag="Mon" zeit="36" hsaal=""/>
191       <Pflicht VoID="250083" teil="1" parity="+" tag="Mon" zeit="32" hsaal=""/>
192       <Pflicht VoID="250084" teil="1" parity="-" tag="" zeit="" hsaal="C206"/>
193     </Pflichten>
194     <TAAST value="250081"/>
195     <TZSZ value=""/>
196     <TA value="250081"/>
197     <Nachfolger>
198       <Paar VoID1="250084" teil1="1" VoID2="250082" teil2="1"/>
199     </Nachfolger>
200     <Spaeter>
201       <Paar VoID1="250082" teil1="1" VoID2="250082" teil2="2"/>
202     </Spaeter>
203     <SelbeZeiten>
204       <SelbeZeit VoID1="250081" teil1="1" VoID2="250081" teil2="2"
205         tag="-1" zeit="0" hsaal="0"/>
206     </SelbeZeiten>
207   </Vorlesungen>
208   <Pausen>
209     <Pause pID="1" value="0"/>
210     <Pause pID="2" value="0"/>
211     <Pause pID="3" value="0"/>
212   </Pausen>
213 </data>
```

7.4 Daten von WS2005/06

Dies sind die Daten, welche für einen realitätsnahen Test verwendet wurden. Sie sind denen aus dem Wintersemester 2005/06 sehr ähnlich.

Listing 7.4: Beispieldaten für WS2005/06

```
Tage = { Mon Die Mit Don Fre };
modul = 2;
splitSmallLarge = 50;
SoftC_Thld_max = 1;
SoftC_Thld_cnt = 0;

pause = [0, 15, 15];

TypZeit = {jederzeit, tagsueber, morgens,
vormittags, mittags, nachmittags, spaetertag, abends};

TypZeitZeiten = [
<32 84> <36 76> <32 48> <40 50> <44 52>
<48 72> <56 72> <68 82>];

HSGruppen = { "UZA2" "UZA2M" "UZA4M" "UZA4Sch" "UZA4" "StrdlM"};

HSortswechsel =
[[ 0 0 5 5 5 20 ]
 [ 0 0 5 5 5 20 ]
 [ 5 5 0 5 5 20 ]
 [ 5 5 5 0 5 25 ]
 [ 5 5 5 5 0 30 ]
 [ 30 30 25 25 30 0 ]];

HSaal = {
< "HS1" 150 { "UZA2M" "UZA2"} [< 32 84 > < 32 70 > < 32 55 > < 32 80 > < 00 00 >]>
< "HS2" 150 { "UZA2M" "UZA2"} [< 32 60 > < 32 50 > < 32 60 > < 32 60 > < 60 80 >]>
< "HS3" 70 { "UZA2M" "UZA2"} [< 32 60 > < 32 50 > < 32 60 > < 32 60 > < 60 80 >]>
< "C209" 30 { "UZA4M" "UZA4"} [< 32 84 > < 32 84 > < 32 84 > < 32 84 > < 32 84 >]>
< "D101" 30 { "UZA4M" "UZA4"} [< 32 84 > < 32 84 > < 32 84 > < 32 84 > < 32 84 >]>
< "C207" 26 { "UZA4M" "UZA4"} [< 32 84 > < 32 84 > < 32 84 > < 32 84 > < 32 84 >]>
< "D103" 24 { "UZA4M" "UZA4"} [< 32 84 > < 32 84 > < 32 84 > < 32 84 > < 32 84 >]>
< "D107" 22 { "UZA4M" "UZA4"} [< 32 84 > < 32 84 > < 32 84 > < 32 84 > < 32 84 >]>
< "A101" 20 { "UZA4M" "UZA4"} [< 32 70 > < 32 70 > < 32 70 > < 32 70 > < 32 70 >]>
< "C206" 15 { "UZA4M" "UZA4"} [< 32 84 > < 32 84 > < 32 84 > < 32 84 > < 32 84 >]>
< "C714" 8 { "UZA4M" "UZA4"} [< 32 84 > < 32 84 > < 32 84 > < 32 84 > < 32 84 >]>
< "A109" 8 { "UZA4M" "UZA4"} [< 32 84 > < 32 84 > < 32 84 > < 32 84 > < 32 84 >]>
< "Schmetterer" 30 {"UZA4Sch"} [< 32 84 > < 32 84 > < 32 84 > < 32 84 > < 32 84 >]>
< "HS1Strudlhof" 30 {"StrdlM"} [< 32 84 > < 32 84 > < 32 84 > < 32 84 > < 32 84 >]>
< "HS2Strudlhof" 30 {"StrdlM"} [< 32 84 > < 32 84 > < 32 84 > < 32 84 > < 32 84 >]>
};

MP_zeit = < 48 , 52 >;

Prof = {
< "Rindler" 1 [ < 32, 80 > < 32, 80 > < 32, 80 > < 32, 80 > < 00, 00 >]>
< "Hoermann" 0 [ < 32, 80 > < 32, 80 > < 32, 80 > < 32, 80 > < 00, 00 >]>
< "Schichl" 0 [ < 36, 60 > < 36, 60 > < 36, 60 > < 36, 60 > < 36, 60 >]>
< "Schoissengeier",1,[< 32, 64 > < 32, 64 > < 32, 50 > < 32, 84 > < 40, 60 >]>
< "Summerer" 0 [ < 32, 72 > < 32, 72 > < 00, 00 > < 32, 72 > < 32, 72 >]>
```

```
52 < "Fischer" 0 [ < 32, 80 > < 32, 80 > < 32, 80 > < 32, 80 > < 32, 80 >]>
53 < "Grosser" 0 [ < 36, 64 > < 36, 64 > < 30, 70 > < 36, 64 > < 36, 42 >]>
54 < "Mitsch" 0 [ < 32, 80 > < 32, 80 > < 32, 80 > < 32, 80 > < 32, 80 >]>
55 < "Stoll" 0 [ < 50, 80 > < 32, 80 > < 32, 70 > < 00, 00 > < 55, 80 >]>
56 < "Huyer" 0 [ < 50, 70 > < 50, 70 > < 50, 70 > < 00, 00 > < 50, 80 >]>
57 < "Mahnkopf" 0 [ < 00, 00 > < 32, 80 > < 32, 80 > < 32, 80 > < 00, 00 >]>
58 < "Krattenthaler", 1 , [ < 40, 74 > < 44, 60 > < 50, 70 > < 00, 00 > < 00, 00 >]>
59 < "Eisenkoelbl", 0, [ < 44, 70 > < 00, 00 > < 00, 00 > < 44, 60 > < 00, 00 >]>
60 < "Fulmek", 0 , [ < 32, 80 > < 32, 80 > < 32, 80 > < 32, 80 > < 32, 80 >]>
61 < "Steinbauer" 0 [ < 32, 80 > < 32, 80 > < 32, 80 > < 32, 52 > < 32, 72 >]>
62 < "Kunzinger" 0 [ < 32, 80 > < 32, 70 > < 32, 80 > < 32, 74 > < 32, 72 >]>
63 < "Teschl" 0 [ < 32, 80 > < 32, 70 > < 00, 00 > < 32, 70 > < 36, 60 >]>
64 < "Haller" 0 [ < 32, 80 > < 32, 70 > < 32, 80 > < 32, 72 > < 32, 60 >]>
65 < "Ulovec" 0 [ < 32, 80 > < 32, 70 > < 36, 80 > < 32, 52 > < 32, 60 >]>
66 < "Raith" 0 [ < 32, 84 > < 32, 84 > < 36, 84 > < 32, 84 > < 00, 00 >]>
67 < "Neumaier" 0 [ < 32, 70 > < 32, 70 > < 36, 70 > < 00, 00 > < 32, 60 >]>
68 < "Rainer" 0 [ < 32, 80 > < 52, 80 > < 50, 80 > < 50, 80 > < 32, 60 >]>
69 < "Baxa" 0 [ < 32 80 > < 00, 00 > < 50, 84 > < 40, 84 > < 32, 60 >]>
70 < "Muthsam" 1 [ < 32 60 > < 32, 50 > < 32, 50 > < 36, 60 > < 00, 00 >]>
71 < "Schmitt" 0 [ < 32 60 > < 32, 60 > < 32, 60 > < 36, 60 > < 32, 60 >]>
72 < "Schlosser" 0 [ < 32, 72 > < 32, 72 > < 32, 72 > < 36, 72 > < 32, 72 >]>
73 < "Ploss" 0 [ < 32, 72 > < 32, 72 > < 32, 72 > < 36, 72 > < 32, 60 >]>
74 < "Greschonig" 0 [ < 32, 60 > < 32, 50 > < 32, 50 > < 36, 60 > < 32, 60 >]>
75 < "Kellner" 0 [ < 40, 84 > < 40, 84 > < 40, 84 > < 40, 84 > < 00, 00 >]>
76 < "Mildenberger" 0 [ < 40, 84 > < 40, 84 > < 40, 84 > < 40, 84 > < 00, 00 >]>
77 < "Feichtinger" 0 [ < 32 80 > < 32, 70 > < 36, 80 > < 32, 52 > < 00, 00 >]>
78 < "Hoffmann-Ostenhof" 0 [ < 40 80 > < 40, 70 > < 40, 80 > < 40, 52 > < 00, 00 >]>
79 < "Auinger" 0 [ < 40, 80 > < 40 70 > < 40, 80 > < 40, 52 > < 40, 60 >]>
80 < "Schmeiser" 0 [ < 40, 70 > < 36 70 > < 36, 80 > < 36, 52 > < 36, 60 >]>
81 < "Fellner" 0 [ < 40, 80 > < 40, 70 > < 40, 80 > < 40, 52 > < 40, 60 >]>
82 < "Markowich" 0 [ < 40, 80 > < 40, 70 > < 40, 52 > < 40, 80 > < 00, 00 >]>
83 < "Mauser" 0 [ < 40, 52 > < 40, 70 > < 40, 80 > < 40, 80 > < 00, 00 >]>
84 < "Haslinger" 1 [ < 40, 80 > < 40, 70 > < 40, 80 > < 40, 52 > < 40, 60 >]>
85 < "Buerger" 0 [ < 40, 80 > < 40, 80 > < 40, 80 > < 40, 62 > < 40, 60 >]>
86 < "Sigmund" 1 [ < 40, 80 > < 40, 80 > < 40, 80 > < 40, 62 > < 40, 60 >]>
87 < "Schmidt" 0 [ < 40, 80 > < 40, 80 > < 40, 80 > < 40, 62 > < 40, 60 >]>
88 < "Kriegl" 0 [ < 40, 80 > < 40, 80 > < 40, 80 > < 40, 62 > < 40, 60 >]>
89 < "F.Hofbauer" 1 [ < 40, 80 > < 40, 80 > < 40, 80 > < 40, 62 > < 40, 60 >]>
90 < "Burde" 0 [ < 40, 70 > < 40, 70 > < 40, 70 > < 40, 72 > < 00, 00 >]>
91 < "Capasso" 0 [ < 40 76 > < 40, 76 > < 40, 76 > < 40, 76 > < 00, 00 >]>
92 < "Cigler" 0 [ <40 65> <40 65> <40 65> <40 65> <00 00> ] >
93 < "Matevosyan" 0 [ <40 65> <40 65> <40 65> <40 65> <00 00> ] >
94 < "Losert" 1 [ < 40, 70 > < 40, 70 > < 40, 70 > < 40, 62 > < 00, 00 >]>
95 < "Komech" 0 [ <40 65> <40 65> <40 65> <40 65> <00 00> ] >
96 < "Neretin" 0 [ <40 70> <40 70> <40 70> <40 70> <00 00> ] >
97 < "Michor" 0 [ <36 72> <36 72> <36 72> <36 72> <36 72> ] >
98 < "Tanzberger" 0 [ < 40, 80 > < 40, 80 > < 40, 80 > < 40, 62 > < 40, 60 >]>
99 < "Humenberger" 0 [ <36 70> <36 70> <36 70> <36 70> <00 00> ] >
100 < "Dorfmayr" 0 [ <36 70> <36 70> <36 70> <36 70> <00 00> ] >
101 < "Goetz" 0 [ <36 70> <36 70> <36 70> <36 70> <00 00> ] >
102 < "Koth" 0 [ <36 84> <36 84> <36 84> <36 84> <00 00> ] >
103 < "Malle" 0 [ <36 84> <36 84> <36 84> <36 84> <00 00> ] >
104 < "Hauer-Typpelt" 0 [ <36 70> <36 70> <36 70> <36 70> <00 00> ] >
105 < "Hejtmanek" 0 [ <40 70> <40 70> <40 70> <40 70> <00 00> ] >
106 < "Groechenig" 0 [ <40 70> <40 70> <40 70> <40 70> <00 00> ] >
107 < "Cap" 0 [ <40 70> <40 70> <40 70> <40 70> <40 64> ] >
108 < "Ruppert" 0 [ <40 70> <40 70> <40 70> <40 70> <40 64> ] >
109 };
```

```
110
111 TypVO = { frei, AnfAllg, Anf1, Anf2, PSA1, PSA2,
112 FortAllg, FortSEAnalysis, FortAlgebra, FortAnalysis, FortNumOpt,
113 FortBio, FortStatistik, FortDiffGeo, FortKombi
114 LAK, PSLAK, LAK2, Abend, Ph, PSPh};
115
116 ueberschneid = // TypVO X TypVO = 1 wenn keine ueb. sein darf
117 [[ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]
118  [ 0 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ]
119  [ 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ]
120  [ 0 1 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]
121  [ 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ]
122  [ 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 ]
123  [ 0 0 0 0 0 0 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 ]
124  [ 0 0 0 0 0 0 1 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 ]
125  [ 0 0 0 0 0 0 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 ]
126  [ 0 0 0 0 0 0 1 1 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 ]
127  [ 0 0 0 0 0 0 1 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 ]
128  [ 0 0 0 0 0 0 1 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 ]
129  [ 0 0 0 0 0 0 1 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 ]
130  [ 0 0 0 0 0 0 1 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 ]
131  [ 0 0 0 0 0 0 1 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 ]
132  [ 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0 0 0 0 ]
133  [ 0 0 0 0 1 1 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 ]
134  [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 0 ]
135  [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 0 ]
136  [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 1 ]
137  [ 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 0 ]];
138
139 VOsetB = {
140 < < "250024" 1 > "Hoermann", 45 "Analysis 1" { "B","C"}
141       100 { "UZA2M" } Anf1 morgens 0>
142 < < "250024" 2 > "Hoermann", 45 "Analysis 1" { "B","C"}
143       100 { "UZA2M" } Anf1 morgens 0>
144 < < "250024" 3 > "Hoermann", 45 "Analysis 1" { "B","C"}
145       100 { "UZA2M" } Anf1 morgens 0>
146 < < "250024" 4 > "Hoermann", 45 "Analysis 1" { "B","C"}
147       100 { "UZA2M" } Anf1 morgens,0>
148 < < "250081" 1 > "Steinbauer" 90 "PS Analysis 1",{"B","C"}
149       20 { "UZA2M" "UZA4M" } PSA1 tagsueber 0>
150 < < "250081" 2 > "Schichl" 90 "PS Analysis 1",{"B","C"}
151       20,{ "UZA2M" "UZA4M" },PSA1,tagsueber,0>
152 < < "250081" 3 > "Grosser" 90 "PS Analysis 1",{"B","C"}
153       20,{ "UZA2M" "UZA4M" },PSA1,tagsueber,0>
154 < < "250081" 4 > "Hoermann" 90 "PS Analysis 1",{"B","C"}
155       20,{ "UZA2M" "UZA4M" },PSA1,tagsueber,0>
156 < < "250081" 5 > "Kunzinger" 90 "PS Analysis 1" {"B","C"}
157       20,{ "UZA2M" "UZA4M" } PSA1 nachmittags 0>
158 < < "250081" 6 > "Teschl" 90 "PS Analysis 1",{"B","C"}
159       20,{ "UZA2M" "UZA4M" } PSA1 nachmittags 0>
160 < < "250081" 7 > "Haller" 90 "PS Analysis 1",{"B","C"}
161       20,{ "UZA2M" "UZA4M" } PSA1 tagsueber 0>
162 < < "250102" 1 > "Schoissengeier" 45 "LinAlg1" { "B","C"}
163       100 { "UZA2M" } Anf1 morgens 0>
164 < < "250102" 2 > "Schoissengeier" 45 "LinAlg1" { "B","C"}
165       100 { "UZA2M" } Anf1 morgens 0>
166 < < "250102" 3 > "Schoissengeier" 45 "LinAlg1" { "B","C"}
167       100 { "UZA2M" } Anf1 morgens 0>
```

```
168 < < "250102" 4 > "Schoissengeier" 45 "LinAlg1" { "B","C"}
169       100 { "UZA2M" } Anf1 morgens 0>
170 < < "250103", 1 > , "Schoissengeier" 90 "PSLinAlg1", { "B","C"}
171       20, { "UZA2M" "UZA4M" }, PSA1, tagsueber, 0>
172 < < "250103", 2 > , "Summerer" 90 "PSLinAlg1", { "B","C"}
173       20, { "UZA2M" "UZA4M" }, PSA1, nachmittags, 0>
174 < < "250103", 3 > , "Fischer" 90 "PSLinAlg1", { "B","C"}
175       20, { "UZA2M" "UZA4M" }, PSA1, tagsueber, 0>
176 < < "250103", 4 > , "Fischer" 90 "PSLinAlg1", { "B","C"}
177       20, { "UZA2M" "UZA4M" }, PSA1, tagsueber, 0>
178 < < "250103", 5 > , "Grosser" 90 "PSLinAlg1", { "B","C"}
179       20, { "UZA2M" "UZA4M" }, PSA1, tagsueber, 0>
180 < < "250103", 6 > , "Summerer" 90 "PSLinAlg1", { "B","C"}
181       20, { "UZA2M" "UZA4M" }, PSA1, tagsueber, 0>
182 < < "250061", 1 > , "Mitsch" 45 "Analysis 3", { "B","C"}
183       50, { "UZA2M" "UZA4M" }, Anf2, morgens, 0>
184 < < "250061", 2 > , "Mitsch" 45 "Analysis 3", { "B","C"}
185       50, { "UZA2M" "UZA4M" }, Anf2, morgens, 0 >
186 < < "250061", 3 > , "Mitsch" 45 "Analysis 3", { "B","C"}
187       50, { "UZA2M" "UZA4M" }, Anf2, morgens, 0 >
188 < < "250061", 4 > , "Mitsch" 45 "Analysis 3", { "B","C"}
189       50, { "UZA2M" "UZA4M" }, Anf2, morgens, 0 >
190 < < "250062", 1 > , "Stoll" 90 "PSAnalysis3", { "B","C"}
191       10, { "UZA2M" "UZA4M" }, PSA2, tagsueber, 0 >
192 < < "250062", 2 > , "Huyer" 90 "PSAnalysis3", { "B","C"}
193       10, { "UZA2M" "UZA4M" }, PSA2, tagsueber, 0 >
194 < < "250062", 3 > , "Stoll" 90 "PSAnalysis3", { "B","C"}
195       10, { "UZA2M" "UZA4M" }, PSA2, tagsueber, 0 >
196 < < "250062", 4 > , "Mahnkopf" 90 "PSAnalysis3", { "B","C"}
197       10, { "UZA2M" "UZA4M" }, PSA2, tagsueber, 0 >
198 < < "250074", 1 > , "Krattenthaler" 90 "Diskrete Mathem.", {"A","B","C"}
199       40, { "UZA2M" "UZA4M" },Anf2,tagsueber,0>
200 < < "250044", 1 > , "Eisenkoelbl" 45 "PS Diskrete Math.", {"A","B","C"}
201       15, { "UZA2M" "UZA4M" },Anf2,tagsueber, 0>
202 < < "250044", 2 > , "Fulmek" 45 "PS Diskrete Math. 2", {"A","B","C"}
203       15,{ "UZA2M" "UZA4M" },Anf2,tagsueber, 0 >
204 < < "250040", 1 > , "Steinbauer" 90 "Grundb. d. Topologie", {"A","B","C"}
205       40,{ "UZA2M" "UZA4M" },Anf2,tagsueber,0>
206 < < "250077", 1 > , "Steinbauer" 45 "PS Grundb. d. Topologie", {"A","B","C"}
207       20,{ "UZA2M" "UZA4M" },PSA2,tagsueber,0>
208 < < "250077", 2 > , "Kunzinger" 45 "PS Grundb. d. Topologie", {"A","B","C"}
209       20,{ "UZA2M" "UZA4M" },PSA2,spaetertag,0>
210 < < "250059", 1 > , "Rindler" 90 "Zahlentheorie", {"A"}
211       30,{ "UZA2M" "UZA4M" },Anf2,morgens,0>
212 < < "250060", 1 > , "Eisenkoelbl" 45 "PS Zahlenth.",{"A"}
213       15,{ "UZA2M" "UZA4M" },Anf2,tagsueber,0>
214 < < "250060", 2 > , "Eisenkoelbl" 45 "PS Zahlenth.",{"A"}
215       15,{ "UZA2M" "UZA4M" },Anf2,tagsueber,0>
216 < < "250073", 1 > , "Steinbauer" 90 "Repit. Analysis 1",{"B","C"}
217       10,{ "UZA2M" "UZA4M" },Anf1,nachmittags,0>
218 < < "250042", 1 > , "Schoissengeier" 90 "Repit. LinAlg1",{"B","C"}
219       10,{ "UZA2M" "UZA4M" },Anf1,nachmittags,0>
220 < < "250063", 1 > , "Mitsch" 90 "Repit. Analysis 3",{"B","C"}
221       10,{ "UZA2M" "UZA4M" },Anf2,abends,0>
222 < < "250012", 2 > , "Ulovec" 90 "Einf. computergest. Arbeiten",{"A","B","C"}
223       30,{ "UZA2M" "UZA4M" },AnfAllg,spaetertag,1>
224
225 //abend:
```

```
226 < < "250057" 1 > "Raith" 90 "Analysis 2 Abend",{"A","B","C"}
227       30,{ "UZA2M" "UZA4M" },Abend,abends,0>
228 < < "250057" 2 > "Raith" 90 "Analysis 2 Abend",{"A","B","C"}
229       30,{ "UZA2M" "UZA4M" },Abend,abends,0>
230 < < "250058" 1 > "Raith" 90 "PS Analysis 2 Abend",{"A" "B" "C"}
231       10,{ "UZA2M" "UZA4M" },Abend,abends,0>
232 < < "250058" 2 > "Rainer" 90 "PS Analysis 2 Abend",{"A" "B" "C"}
233       10,{ "UZA2M" "UZA4M" },Abend,nachmittags,0>
234 < < "250093" 1 > "Raith" 90 "Rep Analysis 2 Abend",{"A" "B" "C"}
235       10,{ "UZA2M" "UZA4M" },Abend,nachmittags,0>
236 < < "250043" 1 > "Baxa" 90 "LinAlg 2 Abend" {"A" "B" "C"}
237       30 { "UZA2M" "UZA4M" } Abend abends 0>
238 < < "250043" 2 > "Baxa" 90 "LinAlg 2 Abend" {"A" "B" "C"}
239       30 { "UZA2M" "UZA4M" } Abend abends 0>
240 < < "250065" 1 > "Baxa" 90 "PS LinAlg 2 Abend" {"A" "B" "C"}
241       15 { "UZA2M" "UZA4M" } Abend nachmittags 0>
242 < < "250096" 1 > "Baxa" 90 "Rep. LinAlg 2 Abend" {"A" "B" "C"}
243       5 { "UZA2M" "UZA4M" } Abend nachmittags 0>
244
245 // physik
246 < < "260224" 1 > "Muthsam" 90 "Analysis f Physik 1" {"A" "B" "C"}
247       30 { "StrdlM"} Ph morgens 0>
248 < < "260224" 2 > "Muthsam" 90 "Analysis f Physik 1" {"A" "B" "C"}
249       30 { "StrdlM"} Ph morgens 0>
250 < < "260224" 3 > "Muthsam" 90 "Analysis f Physik 1" {"A" "B" "C"}
251       30 { "StrdlM"} Ph morgens 0>
252 < < "260225" 1 > "Muthsam" 90 "PS analysis f Physik 1" {"A" "B" "C"}
253       10 { "StrdlM"} PSPh jederzeit 0>
254 < < "260225" 2 > "Muthsam" 90 "PS analysis f Physik 1" {"A" "B" "C"}
255       10 { "StrdlM"} PSPh jederzeit 0>
256 < < "260225" 3 > "Huyer" 90 "PS analysis f Physik 1" {"A" "B" "C"}
257       10 { "StrdlM"} PSPh tagsueber 0>
258 < < "260225" 4 > "Greschonig" 90 "PS analysis f Physik 1" {"A" "B" "C"}
259       10 { "StrdlM"} PSPh tagsueber 0>
260 < < "260225" 5 > "Greschonig" 90 "PS analysis f Physik 1" {"A" "B" "C"}
261       10 { "StrdlM"} PSPh tagsueber 0>
262 < < "260259" 1 > "Muthsam" 90 "Rep. Analysis f Physik 1" {"A" "B" "C"}
263       20 { "StrdlM"} Ph tagsueber 0>
264 < < "260226" 1 > "Schmitt" 75 "LinAlg Physik" {"A" "B" "C"}
265       30 { "StrdlM"} Ph morgens 0>
266 < < "260226" 2 > "Schmitt" 75 "LinAlg Physik" {"A" "B" "C"}
267       30 { "StrdlM"} Ph morgens 0>
268 < < "260227" 1 > "Schmitt" 90 "PS LinAlg Physik" {"A" "B" "C"}
269       10 { "StrdlM"} PSPh jederzeit 0>
270 < < "260227" 2 > "Schlosser" 90 "PS LinAlg Physik" {"A" "B" "C"}
271       10 { "StrdlM"} PSPh tagsueber 0>
272 < < "260227" 3 > "Schlosser" 90 "PS LinAlg Physik" {"A" "B" "C"}
273       10 { "StrdlM"} PSPh tagsueber 0>
274 < < "260227" 4 > "Ploss" 90 "PS LinAlg Physik" {"A" "B" "C"}
275       10 { "StrdlM"} PSPh tagsueber 0>
276 < < "260227" 5 > "Ploss" 90 "PS LinAlg Physik" {"A" "B" "C"}
277       10 { "StrdlM"} PSPh tagsueber 0>
278 < < "260016" 1 > "Schmitt" 90 "Rep LinAlg Physik" {"A" "B" "C"}
279       10 { "StrdlM"} Ph nachmittags 0>
280
281 //2ter abschnitt
282 < < "250108" 1 > "Summerer" 90 "Algebra 2" {"A" "B" "C"}
283       50 { "UZA2M" "UZA4M" } FortAllg vormittags 0>
```

```
< < "250109" 1 > "Summerer" 45 "PS Algebra 2" {"A" "B" "C"}
     25 { "UZA2M" "UZA4M" } FortAllg tagsueber 0>
< < "250150" 1 > "Kellner" 90 "Grundb mathm Logik" {"A" "B" "C"}
     30 { "UZA2M" "UZA4M" } FortAllg tagsueber 0>
< < "250151" 1 > "Mildenberger" 45 "PS Grundb mathm Logik" {"A" "B" "C"}
     10 { "UZA2M" "UZA4M" } FortAllg tagsueber 0>
< < "250165" 1 > "Feichtinger" 45 "Funktionalanalysis 1" {"A" "B" "C"}
     20 { "UZA2M" "UZA4M" } FortAllg tagsueber 0>
< < "250165" 2 > "Feichtinger" 45 "Funktionalanalysis 1" {"A" "B" "C"}
     20 { "UZA2M" "UZA4M" } FortAllg tagsueber 0>
< < "250165" 3 > "Feichtinger" 45 "Funktionalanalysis 1" {"A" "B" "C"}
     20 { "UZA2M" "UZA4M" } FortAllg tagsueber 0>
< < "250112" 1 > "Feichtinger" 45 "PS Funktionalanalysis 1" {"A" "B" "C"}
     10 { "UZA2M" "UZA4M" } FortAllg tagsueber 0>
< < "250022" 1 > "Hoffmann-Ostenhof" 45 "PDE 1" {"A" "B" "C"}
     20 { "UZA2M" "UZA4M" } FortAllg tagsueber 0>
< < "250022" 2 > "Hoffmann-Ostenhof" 45 "PDE 1" {"A" "B" "C"}
     20 { "UZA2M" "UZA4M" } FortAllg tagsueber 0>
< < "250022" 3 > "Hoffmann-Ostenhof" 45 "PDE 1" {"A" "B" "C"}
     20 { "UZA2M" "UZA4M" } FortAllg tagsueber 0>
< < "250120" 1 > "Hoffmann-Ostenhof" 45 "PS PDE 1" {"A" "B" "C"}
     10 { "UZA2M" "UZA4M" } FortAllg tagsueber 0>
< < "250030" 1 > "Schmitt" 90 "Geschichte d Mathem." {"A" "B" "C"}
     20 { "UZA2M" "UZA4M" } FortAllg tagsueber 0>
< < "250164" 1 > "Auinger" 90 "Gruppentheorie" {"A" "B" "C"}
     10 { "UZA2M" "UZA4M" } FortAlgebra tagsueber 0>
< < "250164" 2 > "Auinger" 90 "Gruppentheorie" {"A" "B" "C"}
     10 { "UZA2M" "UZA4M" } FortAlgebra tagsueber 0>
< < "250166" 1 > "Auinger" 90 "PS Gruppenth." {"A" "B" "C"}
     10 { "UZA2M" "UZA4M" } FortAlgebra tagsueber 0>
< < "250134" 1 > "Schmeiser" 90 "Gew DGL 2" {"A" "B" "C"}
     20 { "UZA2M" "UZA4M" } FortAnalysis tagsueber 0>
< < "250095" 1 > "Fellner" 90 "PS Gew DGL 2" {"A" "B" "C"}
     20 { "UZA2M" "UZA4M" } FortAnalysis tagsueber 0>
< < "250114" 1 > "Haslinger" 45 "Komplexe Analysis 2" {"A" "B" "C"}
     20 { "UZA2M" "UZA4M" } FortAnalysis tagsueber 0>
< < "250114" 2 > "Haslinger" 45 "Komplexe Analysis 2" {"A" "B" "C"}
     20 { "UZA2M" "UZA4M" } FortAnalysis tagsueber 0>
< < "250115" 1 > "Haslinger" 45 "PS Komplexe Analysis 2" {"A" "B" "C"}
     10 { "UZA2M" "UZA4M" } FortAnalysis tagsueber 0>
< < "250126" 1 > "Schichl" 45 "Numerische Math. 2" {"A" "B" "C"}
     15 { "UZA2M" "UZA4M" } FortNumOpt tagsueber 1>
< < "250126" 2 > "Schichl" 90 "Numerische Math. 2" {"A" "B" "C"}
     15 { "UZA2M" "UZA4M" } FortNumOpt tagsueber 1>
< < "250127" 1 > "Schichl" 45 "PS Numerische Math. 2" {"A" "B" "C"}
     10 { "UZA2M" "UZA4M" } FortNumOpt tagsueber 1>
< < "250038" 1 > "Neumaier" 45 "Optimierung Variat." {"A" "B" "C"}
     20 { "UZA2M" "UZA4M" } FortNumOpt tagsueber 0>
< < "250038" 2 > "Neumaier" 45 "Optimierung Variat." {"A" "B" "C"}
     20 { "UZA2M" "UZA4M" } FortNumOpt tagsueber 0>
< < "250038" 3 > "Neumaier" 45 "Optimierung Variat." {"A" "B" "C"}
     20 { "UZA2M" "UZA4M" } FortNumOpt tagsueber 0>
< < "250110" 1 > "metaBuergerSigmund" 90 "Biomathematik" {"A" "B" "C"}
     15 { "UZA2M" "UZA4M" } FortBio tagsueber 0>
< < "250110" 2 > "metaBuergerSigmund" 90 "Biomathematik" {"A" "B" "C"}
     15 { "UZA2M" "UZA4M" } FortBio tagsueber 0>
< < "250111" 1 > "Buerger" 45 "PS Biomathematik" { "A" "B" "C" }
     5 { "UZA2M" "UZA4M" } FortBio tagsueber 0>
```

```
342 < < "250008" 1 > "Schmidt" 90 "Stoch Prozesse" { "A" "B" "C" }
343       5 { "UZA2M" "UZA4M" } FortStatistik tagsueber 0>
344 < < "250008" 2 > "Schmidt" 90 "Stoch Prozesse" { "A" "B" "C" }
345       5 { "UZA2M" "UZA4M" } FortStatistik tagsueber 0>
346 < < "250064" 1 > "Kriegl" 45 "DiffGeo 2" { "A" "B" "C" }
347       5 { "UZA2M" "UZA4M" } FortDiffGeo tagsueber 0>
348 < < "250064" 2 > "Kriegl" 45 "DiffGeo 2" { "A" "B" "C" }
349       5 { "UZA2M" "UZA4M" } FortDiffGeo tagsueber 0>
350 < < "250064" 3 > "Kriegl" 45 "DiffGeo 2" { "A" "B" "C" }
351       5 { "UZA2M" "UZA4M" } FortDiffGeo tagsueber 0>
352 < < "250078" 1 > "Kriegl" 90 "PS DiffGeo 2" {"A" "B" "C"}
353       5 { "UZA2M" "UZA4M" } FortDiffGeo tagsueber 0>
354 < < "250118" 1 > "F.Hofbauer" 90 "Hoeh Wahrscheinlkth" {"A" "B" "C"}
355       10 { "UZA2M" "UZA4M" } FortStatistik tagsueber 0>
356 < < "250118" 2 > "F.Hofbauer" 90 "Hoeh Wahrscheinlkth" {"A" "B" "C"}
357       10 { "UZA2M" "UZA4M" } FortStatistik tagsueber 0>
358 < < "250104" 1 > "Burde" 90 "Analytische ZT" {"A" "B" "C"}
359       10 { "UZA2M" "UZA4M" } FortAlgebra nachmittags 0>
360 < < "250104" 2 > "Burde" 90 "Analytische ZT" {"A" "B" "C"}
361       10 { "UZA2M" "UZA4M" } FortAlgebra nachmittags 0>
362 < < "250124" 1 > "Mahnkopf" 90 "Alg Geometrie" {"A" "B" "C"}
363       5 { "UZA2M" "UZA4M" } FortAlgebra tagsueber 0>
364 < < "250124" 2 > "Mahnkopf" 90 "Alg Geometrie" {"A" "B" "C"}
365       5 { "UZA2M" "UZA4M" } FortAlgebra tagsueber 0>
366 < < "250113" 1 > "Grosser" 90 "Colombeau Alg" {"A" "B" "C"}
367       5 { "UZA2M" "UZA4M" } FortAnalysis tagsueber 0>
368 < < "250113" 2 > "Grosser" 90 "Colombeau Alg" {"A" "B" "C"}
369       5 { "UZA2M" "UZA4M" } FortAnalysis tagsueber 0>
370 < < "250014" 1 > "Neumaier" 90 "math Methoden d Datenanalyse" {"A" "B" "C"}
371       10 { "UZA2M" "UZA4M" } FortNumOpt tagsueber 0>
372 < < "250014" 2 > "Neumaier" 90 "math Methoden d Datenanalyse" {"A" "B" "C"}
373       10 { "UZA2M" "UZA4M" } FortNumOpt tagsueber 0>
374 < < "250009" 1 > "Sigmund" 90 "Einf Spieltheorie" {"A" "B" "C"}
375       15 { "UZA2M" "UZA4M" } FortBio tagsueber 0>
376 < < "250168" 1 > "Capasso" 90 "Populationsdyn v Infektionskrankh" {"A" "B" "C" }
377       10 { "UZA2M" "UZA4M" } FortBio abends 0>
378 < < "250141" 1 > "Rindler" 45 "AK ZT" { "C" }
379       30 { "UZA2M" "UZA4M" } FortAlgebra morgens 0>
380 < < "250141" 2 > "Rindler" 45 "AK ZT" { "C" }
381       30 { "UZA2M" "UZA4M" } FortAlgebra morgens 0>
382 < < "250141" 3 > "Rindler" 45 "AK ZT" { "C" }
383       30 { "UZA2M" "UZA4M" } FortAlgebra morgens 0>
384 < < "250141" 4 > "Rindler" 45 "AK ZT" { "C" }
385       30 { "UZA2M" "UZA4M" } FortAlgebra morgens 0>
386 < < "250100" 1 > "Eisenkoelbl" 45 "PS AK ZT" {"C"}
387       60 { "UZA2M" "UZA4M" } FortAlgebra tagsueber 0>
388 < < "250075" 1 > "Cigler" 90 "AK Kombinatorik" {"A" "B" "C"}
389       10 { "UZA2M" "UZA4M" } FortKombi spaetertag 0>
390 < < "250123" 1 > "Matevosyan" 90 "AK PDE freie RWP" {"A" "B" "C"}
391       5 { "UZA2M" "UZA4M" } FortAnalysis tagsueber 0>
392 < < "250137" 1 > "Losert" 90 "AK Moderne Analysis" {"A" "B" "C"}
393       10 { "UZA2M" "UZA4M" } FortAnalysis tagsueber 0>
394 < < "250137" 2 > "Losert" 90 "AK Moderne Analysis" {"A" "B" "C"}
395       10 { "UZA2M" "UZA4M" } FortAnalysis tagsueber 0>
396 < < "250167" 1 > "Komech" 90 "Dispersion Attractors .. KleinGordon" {"A" "B" "C"}
397       5 { "UZA2M" "UZA4M" } FortAnalysis tagsueber 0>
398 < < "250167" 2 > "Komech" 90 "Dispersion Attractors .. KleinGordon" {"A" "B" "C"}
399       5 { "UZA2M" "UZA4M" } FortAnalysis tagsueber 0>
```

```
400 < < "250099" 1 > "Neretin" 90 "AK DiffGeo Matrix+classGroups" {"B" "C"}
401       5 { "UZA2M" "UZA4M" } FortDiffGeo tagsueber 0>
402 < < "250099" 2 > "Neretin" 90 "AK DiffGeo Matrix+classGroups" {"B" "C"}
403       5 { "UZA2M" "UZA4M" } FortDiffGeo tagsueber 0>
404 < < "250125" 1 > "Michor" 90 "AK DiffGeo sympl Geo" {"A" "B" "C"}
405       7 { "UZA2M" "UZA4M" } FortDiffGeo tagsueber 0>
406 < < "250125" 2 > "Michor" 90 "AK DiffGeo sympl Geo" {"A" "B" "C"}
407       7 { "UZA2M" "UZA4M" } FortDiffGeo tagsueber 0>
408 < < "250105" 2 > "Haller" 90 "AK Topologie DiffTopo" {"A" "B" "C"}
409       5 { "UZA2M" "UZA4M" } FortDiffGeo tagsueber 0>
410 < < "250117" 2 > "F.Hofbauer" 90 "AK WahrscheinlchkTH" {"A" "B" "C"}
411       8 { "UZA2M" "UZA4M" } FortStatistik tagsueber 0>
412 // seminare
413 < < "250027" 1 > "Rindler" 180 "KO mathem 'Kolloquium" {"A" "B" "C" }
414       20 { "UZA4M" } FortAllg abends 0>
415 < < "250066" 1 > "metaBaxaSchoissengeier" 90 "SE Zahlentheorie" {"A" "B" "C"}
416       5 { "UZA2M" "UZA4M" } FortAlgebra tagsueber 0>
417 < < "250131" 1 > "Krattenthaler" 90 "SE Kombinatorik" {"A" "B" "C"}
418       5 { "UZA2M" "UZA4M" } FortKombi tagsueber 0>
419 < < "250019" 1 > "metaMarkowichMauser" 90 "SE DiffGleichungen" {"A" "B" "C"}
420       5 { "UZA2M" "UZA4M" } FortSEAnalysis tagsueber 0>
421 < < "250020" 1 > "metaMarkowichMauser" 90 "SE Angew Nichtlin Analysis" {"A" "B" "C"}
422       5 { "UZA2M" "UZA4M" } FortSEAnalysis tagsueber 0>
423 < < "250116" 1 > "Haslinger" 90 "SE komplexe Analysis" { "A" "B" "C"}
424       5 { "UZA2M" "UZA4M" } FortSEAnalysis tagsueber 0>
425 < < "250121" 1 > "Hoermann" 90 "SE Pseudodiff Operatoren" { "A" "B" "C"}
426       5 { "UZA2M" "UZA4M" } FortSEAnalysis tagsueber 0>
427 < < "250122" 1 > "Kunzinger" 90 "SE nonlinear distributional gemoetry" { "A" "B" "C"}
428       5 { "UZA2M" "UZA4M" } FortSEAnalysis tagsueber 0>
429 // ue:
430 < < "250135" 1 > "Schmeiser" 90 "SE pde in der Zellbiologie" { "A" "B" "C"}
431       5 { "UZA2M" "UZA4M" } FortBio tagsueber 0>
432 < < "250138" 1 > "Losert" 90 "SE harmonische Analyse" { "A" "B" "C"}
433       5 { "UZA2M" "UZA4M" } FortSEAnalysis tagsueber 0>
434 < < "250145" 1 > "Hejtmanek" 90 "SE nonstandard Analysis" {"A" "B" "C"}
435       5 { "UZA4Sch" } FortSEAnalysis tagsueber 0>
436 < < "250171" 1 > "metaFeichtingerGroechenig" 90 "SE moderne Analysis" {"A" "B" "C"}
437       5 { "UZA4Sch" } FortSEAnalysis tagsueber 0>
438 < < "250021" 1 > "Neumaier" 90 "SE mathematik und physik" {"A" "B" "C"}
439       6 { "UZA2M" "UZA4M" } FortNumOpt tagsueber 0>
440 < < "250128" 1 > "metaNeumaierSchichl" 90 "SE Opti" {"A" "B" "C"}
441       6 { "UZA2M" "UZA4M" } FortNumOpt tagsueber 1>
442 < < "250034" 1 > "Buerger" 90 "SE mathem Populationsgenetik" {"A" "B" "C"}
443       6 { "UZA2M" "UZA4M" } FortBio jederzeit 0>
444 < < "250170" 1 > "Capasso" 90 "SE biomathem Modelle" {"A" "B" "C"}
445       6 { "UZA2M" "UZA4M" } FortBio jederzeit 0>
446 //< < "250031" 1 > "metaCapMichorRuppert" 135 "SE LieGruppen" {"A" "B" "C"}
447       6 { "UZA2M" "UZA4M" } FortDiffGeo jederzeit 0>
448
449 // LAK Vorlesungen
450 < < "250169" 1 > "Tanzberger" 90 "Geschlechtsspezif Prob MathUnterr" {"A" "B" "C"}
451       20 { "UZA2M" "UZA4M" } LAK abends 0>
452 < < "250101" 1 > "Kunzinger" 90 "Einf mathem Arbeiten" {"A"}
453       50 { "UZA2M" "UZA4M" } LAK morgens 0>
454 < < "250129" 1 > "Humenberger" 90 "Ang Mathem f LAK" {"A" "B" "C"}
455       50 { "UZA2M" "UZA4M" } LAK2 vormittags 0>
456 < < "250129" 2 > "Humenberger" 45 "Ang Mathem f LAK" {"A" "B" "C"}
457       50 { "UZA2M" "UZA4M" } LAK2 vormittags 0>
```

```
< < "250106" 1 > "Humenberger" 45 "PS Ang Mathem f LAK" {"A" "B" "C"}
     15 { "UZA2M" "UZA4M" } LAK2 tagsueber 0>
< < "250106" 2 > "Dorfmayr" 45 "PS Ang Mathem f LAK" {"A" "B" "C"}
     15 { "UZA2M" "UZA4M" } LAK2 tagsueber 0>
< < "250098" 1 > "Goetz" 90 "Hoehere Analysis f LAK" {"A" "B" "C"}
     40 { "UZA2M" "UZA4M" } LAK2 tagsueber 0>
< < "250098" 2 > "Goetz" 90 "Hoehere Analysis f LAK" {"A" "B" "C"}
     40 { "UZA2M" "UZA4M" } LAK2 tagsueber 0>
< < "250159" 1 > "Goetz" 90 "PS Hoehere Analysis f LAK" {"A" "B" "C"}
     40 { "UZA2M" "UZA4M" } LAK2 tagsueber 0>
< < "250158" 1 > "Mitsch" 90 "SE LAK Algebra" {"A" "B" "C"}
     15 { "UZA2M" "UZA4M" } LAK2 tagsueber 0>
< < "250107" 1 > "Raith" 90 "SE LAK WKTh" {"A" "B" "C"}
     15 { "UZA2M" "UZA4M" } LAK2 tagsueber 0>
< < "250097" 1 > "Goetz" 90 "Einf Fachdidaktik" {"A" "B" "C"}
     30 { "UZA2M" "UZA4M" } LAK2 tagsueber 0>
< < "250157" 1 > "Koth" 90 "SE Schulpraktikum" {"A" "B" "C"}
     30 { "UZA2M" "UZA4M" } LAK2 abends 0>
< < "250050" 1 > "Malle" 90 "SE Unterrichtsplanung" {"A" "B" "C"}
     30 { "UZA2M" "UZA4M" } LAK2 abends 0>
< < "250051" 1 > "Malle" 90 "SE Fachdidaktik" {"A" "B" "C"}
     30 { "UZA2M" "UZA4M" } LAK2 abends 0>
< < "250048" 1 > "Humenberger" 90 "SE Didaktik d Mathem" {"A" "B" "C"}
     30 { "UZA2M" "UZA4M" } LAK2 tagsueber 0>
< < "250163" 1 > "Ulovec" 90 "SE neue Entwickl d Fachdidaktik" {"A" "B" "C"}
     30 { "UZA2M" "UZA4M" } LAK2 tagsueber 0>
< < "250130" 1 > "Humenberger" 90 "Schulmathe 1 Zahlbereiche" {"A" "B" "C"}
     30 { "UZA2M" "UZA4M" } LAK tagsueber 0>
< < "250094" 1 > "Hauer-Typpelt" 45 "PS Schulmathe 1" {"A" "B" "C"}
     30 { "UZA2M" "UZA4M" } LAK tagsueber 0>
< < "250143" 1 > "Koth" 90 "Schulmathe 4 Vektorrech" {"A" "B" "C"}
     30 { "UZA2M" "UZA4M" } LAK2 tagsueber 0>
< < "250144" 1 > "Koth" 45 "PS Schulmathe 4 Vektorrech" {"A" "B" "C"}
     30 { "UZA2M" "UZA4M" } LAK2 tagsueber 0>
< < "250144" 2 > "Koth" 45 "PS Schulmathe 4 Vektorrech" {"A" "B" "C"}
     30 { "UZA2M" "UZA4M" } LAK2 tagsueber 0>
< < "250147" 1 > "Malle" 90 "Schulmathem 6 DiffRech" {"A" "B" "C"}
     30 { "UZA2M" "UZA4M" } LAK2 tagsueber 0>
< < "250148" 1 > "Malle" 45 "PS Schulmathem 6 DiffRech" {"A" "B" "C"}
     30 { "UZA2M" "UZA4M" } LAK2 tagsueber 0>
};

wuenscheZeit = {
< < "250103" 4 > -.2 { < 1 Mon 0 > } >
< < "250081" 5 > 0.2 { < 0 Mon 62 > < 0 Mon 64 > < 0 Mon 66 > < 0 Mon 68 > } >
< < "250081" 6 > 0.2 { < 0 Mon 56 > < 0 Mon 60 > < 0 Mon 64 > } >
< < "250081" 7 > 0.2 { < 0 Mon 56 > < 0 Mon 60 > < 0 Mon 64 > } >
< < "250062" 4 > 0.2 { < 1 Mit 0 > } >
< < "250074" 1 > 0.2 { < 0 Mit 60 > < 0 Mon 56 > < 0 Mon 52 > } >
< < "250062" 1 > 0.3 { < 1 Mit 0 > } >
< < "250062" 3 > 0.3 { < 1 Mit 0 > } >
< < "250044" 1 > 0.3 { < 1 Mon 52 > < 1 Mon 56 > } >
< < "250044" 2 > 0.2 { < 0 Mon 52 > < 0 Mon 56 > } >
< < "250040" 1 > .3 { < 1 Mon 0 > < 1 Mit 0 > } >
< < "250042" 1 > -.3 { < 1 Mit 0 > } >
< < "250081" 1 > -.2 { < 1 Fre 64 > < 1 Fre 60 > } >
< < "250081" 2 > 0.2 { < 1 Fre 0 > < 1 Don 0 > } >
< < "250150" 1 > 0.2 { < 1 Mit 0 > < 1 Mit 44 > } >
```

```
516 < < "250101" 1 > −.9 { < 1 Mon 0 > } >
517 < < "250104" 1 > 0.2 { < 0 Die 60 > < 0 Mon 64 > } >
518 < < "250131" 1 > 0.5 { < 1 Mon 32 > } >
519 < < "250040" 1 > 0.2 { < 0 Mon 36 > < 0 Mon 38 > < 0 Mon 40 > < 0 Mon 42 > } >
520 < < "250158" 1 > −.2 { < 1 Fre 00 > } >
521 < < "250126" 1 > 0.3 { < 1 Mit 48 > } >
522 < < "250126" 2 > 0.3 { < 1 Don 48 > } >
523 < < "250027" 1 > 0.5 { < 1 Mit 0 > } >
524 };
525
526 wuenscheOrt = {
527 < < "250103" 4 > 0.2 { "C207" "D101" } >
528 < < "250040" 1 > 0.3 { "C209" } >
529 < < "250044" 1 > 0.3 { "C209" } >
530 < < "250077" 1 > −.5 { "HS1" "HS2" } >
531 < < "250077" 2 > −.5 { "HS1" "HS2" } >
532 < < "250014" 1 > 0.2 { "C206" } >
533 < < "250123" 1 > 0.2 { "C714" } >
534 < < "250131" 1 > 0.5 { "HS3" } >
535 < < "250169" 1 > 0.2 { "D103" } >
536 < < "250098" 1 > 0.2 { "HS3" } >
537 < < "250019" 1 > 0.2 { "C714" } >
538 < < "250020" 1 > 0.2 { "C714" } >
539 < < "250131" 1 > −.2 { "C714" } >
540 < < "250126" 1 > 0.3 { "D107" } >
541 < < "250126" 2 > 0.3 { "D107" } >
542 < < "250027" 1 > 0.5 { "C209" } >
543 };
544
545 pflichten = {
546  < < "250024" 1 > 1 Mon 32 "HS1" >
547  < < "250102" 1 > 1 Mon 36 "HS1" >
548  < < "250061" 1 > 1 Mon 44 "HS2" >
549  < < "250059" 1 > 1 Mon 32 "none">
550
551 //haupt ps verschieben
552 < < "250081" 5 > 0 Mon 66 "none" >
553 < < "250081" 7 > 0 Mon 68 "none" >
554 < < "250103" 4 > 0 Fre 60 "none" >
555 < < "250103" 5 > 0 Die 64 "none" >
556 < < "250062" 2 > 0 Fre 64 "none" >
557 };
558
559 ausschluss = {
560  < < "250062", 1 > , 0, Mon, 0, "C206" >
561  < < "250062", 2 > , 0, Die, 0, "C206" >
562  < < "250062", 3 > , 0, Mit, 0, "HS1" >
563 };
564
565 nachfolger = {
566 };
567
568 TAAST = {
569  "250103", "250062", "250044", "250077", "250081", "250060" "250058"
570  "260225" "260227" // ph
571 };
572
573 TZSZ = {
```

```
574 "250061" "250102" "250024" "250057" "250043"
575 "250114"
576 "250164" "250022" "250165" "250038" "250110" "250008" "250064"
577 "250104" "250118"
578 "250124" "250014"
579 "250131" "250137" "250167" "250099"
580 "250126"
581 "260226" "260224" // ph
582 "250129" "250098"
583 "250125" //michor sympl geo
584 };
585
586 teileAufsteigend = {
587  "250102" "250024" "250061" "250057" "250043" "260224"
588  "250114" "250164" "250022" "250165" "250038" "250110" "250008" "250064"
589  "250126" "250113"
590  "250104" "250118"
591  "250124" "250014"
592  "250131" "250137" "250167" "250099"
593  "260226" "260224" //ph
594  "250129" "250098"
595  "250125" //michor sympl geo
596 };
597
598 selbeZeit = {
599 < < "250081" 1 > < "250081" 2 > 1 0 0 >
600 < < "250081" 1 > < "250081" 5 > 1 0 0 >
601 < < "250103" 1 > < "250103" 6 > 1 1 0 >
602 < < "250060" 1 > < "250060" 2 > -1 0 0 > // zahlenth. PS
603 < < "250044" 1 > < "250044" 2 > -1 0 0 > // diskrete math. PS
604 < < "250058" 1 > < "250058" 2 > -1 0 0 > // PS ana2 abend
605 < < "250043" 1 > < "250043" 2 > -1 0 0 > // PS linalg 2 abend
606 < < "250077" 1 > < "250077" 2 > -1 0 0 > // PS topologie
607 };
608
609 spaeter = {
610 < < "250150" 1 > < "250151" 1 > >
611 < < "250126" 2 > < "250127" 1 > >
612 };
613
614 hsBelegt = {
615 < "HS1" Don 60 80 >
616 < "HS1" Mit 55 80 >
617 < "HS2" Don 48 60 >
618 < "D101" Mit 48 52 >
619 < "C209" Mon 52 60 >
620 };
621
622 profBelegt = {
623  < "Rindler" Die 50 54 >
624 };
625
626 metaProf = {
627  < "metaBaxaSchoissengeier" { "Baxa" "Schoissengeier" } >
628  < "metaNeumaierSchichl" { "Schichl" "Neumaier" } >
629  < "metaBuergerSigmund" { "Buerger" "Sigmund" } >
630  < "metaMarkowichMauser" { "Markowich" "Mauser" } >
631  < "metaFeichtingerGroechenig" { "Feichtinger" "Groechenig" } >
```

```
632  < "metaCapMichorRuppert" { "Cap" "Michor" "Ruppert" } >
633 };
```

7.5 Lösung zu WS2005/06

Dies ist die aus den Daten berechnete Lösung. Benötigte Lösungszeit ist ungefähr 10 Sekunden. In jeder Zeile ist der Reihe nach die Identifikation der Vorlesung, deren aufsteigende Teilnummer, der Titel, die Identifikation des Vortragenden, der berechnete Tag, die berechnete Uhrzeit (umgewandelt in normales 24-Stunden Format), die Dauer in Minuten und die Identifikation des berechneten Hörsaals zu sehen.

Listing 7.5: Beispieldaten für WS2005/06

```
Feasible Solution with Objective Value: 22.0097

res = {
<"250024",1,"Analysis 1","Hoermann",Mon,8:00,45,"HS1">,
<"250024",2,"Analysis 1","Hoermann",Die,8:00,45,"HS1">,
<"250024",3,"Analysis 1","Hoermann",Mit,8:00,45,"HS1">,
<"250024",4,"Analysis 1","Hoermann",Don,8:00,45,"HS1">,
<"250081",1,"PS Analysis 1","Steinbauer",Mon,10:00,90,"D101">,
<"250081",2,"PS Analysis 1","Schichl",Mon,10:00,90,"C209">,
<"250081",3,"PS Analysis 1","Grosser",Fre,9:00,90,"C209">,
<"250081",4,"PS Analysis 1","Hoermann",Mon,10:00,90,"C207">,
<"250081",5,"PS Analysis 1","Kunzinger",Mon,16:30,90,"D101">,
<"250081",6,"PS Analysis 1","Teschl",Mon,12:00,90,"D101">,
<"250081",7,"PS Analysis 1","Haller",Mon,17:00,90,"C209">,
<"250102",1,"LinAlg1","Schoissengeier",Mon,9:00,45,"HS1">,
<"250102",2,"LinAlg1","Schoissengeier",Die,9:00,45,"HS1">,
<"250102",3,"LinAlg1","Schoissengeier",Mit,9:00,45,"HS1">,
<"250102",4,"LinAlg1","Schoissengeier",Don,9:00,45,"HS1">,
<"250103",1,"PSLinAlg1","Schoissengeier",Mon,10:00,90,"D103">,
<"250103",2,"PSLinAlg1","Summerer",Mon,12:00,90,"C207">,
<"250103",3,"PSLinAlg1","Fischer",Fre,9:00,90,"D101">,
<"250103",4,"PSLinAlg1","Fischer",Mon,15:00,90,"C209">,
<"250103",5,"PSLinAlg1","Grosser",Mit,16:00,90,"C209">,
<"250103",6,"PSLinAlg1","Summerer",Mon,10:00,90,"D107">,
<"250061",1,"Analysis 3","Mitsch",Mon,11:00,45,"HS2">,
<"250061",2,"Analysis 3","Mitsch",Die,11:00,45,"HS2">,
<"250061",3,"Analysis 3","Mitsch",Mit,11:00,45,"HS2">,
<"250061",4,"Analysis 3","Mitsch",Don,11:00,45,"HS2">,
<"250062",1,"PSAnalysis3","Stoll",Die,9:00,90,"C209">,
<"250062",2,"PSAnalysis3","Huyer",Mon,16:00,90,"C207">,
<"250062",3,"PSAnalysis3","Stoll",Mit,9:00,90,"C209">,
<"250062",4,"PSAnalysis3","Mahnkopf",Die,9:00,90,"D101">,
<"250074",1,"Diskrete Mathem.","Krattenthaler",Die,13:00,90,"HS1">,
<"250044",1,"PS Diskrete Math.","Eisenkoelbl",Mon,12:00,45,"D103">,
<"250044",2,"PS Diskrete Math. 2","Fulmek",Don,9:00,45,"C209">,
<"250040",1,"Grundb. d. Topologie","Steinbauer",Mit,12:00,90,"HS1">,
<"250077",1,"PS Grundb. d. Topologie","Steinbauer",Die,9:00,45,"C207">,
<"250077",2,"PS Grundb. d. Topologie","Kunzinger",Mon,14:00,45,"D101">,
<"250059",1,"Zahlentheorie","Rindler",Mon,8:00,90,"C209">,
<"250060",1,"PS Zahlenth.","Eisenkoelbl",Mon,11:00,45,"C209">,
<"250060",2,"PS Zahlenth.","Eisenkoelbl",Don,11:00,45,"C209">,
<"250073",1,"Repit. Analysis 1","Steinbauer",Fre,12:00,90,"A101">,
<"250042",1,"Repit. LinAlg1","Schoissengeier",Don,13:00,90,"D101">,
<"250063",1,"Repit. Analysis 3","Mitsch",Die,17:00,90,"D103">,
<"250012",2,"Einf. computergest. Arbeiten","Ulovec",Mit,14:00,90,"C209">,
<"250057",1,"Analysis 2 Abend","Raith",Mon,18:00,90,"D101">,
<"250057",2,"Analysis 2 Abend","Raith",Die,18:00,90,"D101">,
<"250058",1,"PS Analysis 2 Abend","Raith",Mit,18:30,90,"C209">,
```

```
<"250058",2,"PS Analysis 2 Abend","Rainer",Fre,12:00,90,"D103">,
<"250093",1,"Rep Analysis 2 Abend","Raith",Die,12:00,90,"C207">,
<"250043",1,"LinAlg 2 Abend","Baxa",Mit,17:00,90,"D101">,
<"250043",2,"LinAlg 2 Abend","Baxa",Don,17:00,90,"D101">,
<"250065",1,"PS LinAlg 2 Abend","Baxa",Don,12:00,90,"D107">,
<"250096",1,"Rep. LinAlg 2 Abend","Baxa",Mit,12:30,90,"D103">,
<"260224",1,"Analysis f Physik 1","Muthsam",Mon,10:30,90,"HS1Strudlhof">,
<"260224",2,"Analysis f Physik 1","Muthsam",Die,10:30,90,"HS1Strudlhof">,
<"260224",3,"Analysis f Physik 1","Muthsam",Mit,10:30,90,"HS1Strudlhof">,
<"260225",1,"PS analysis f Physik 1","Muthsam",Mit,8:00,90,"HS2Strudlhof">,
<"260225",2,"PS analysis f Physik 1","Muthsam",Don,9:00,90,"HS2Strudlhof">,
<"260225",3,"PS analysis f Physik 1","Huyer",Mon,12:30,90,"HS1Strudlhof">,
<"260225",4,"PS analysis f Physik 1","Greschonig",Mon,9:00,90,"HS1Strudlhof">,
<"260225",5,"PS analysis f Physik 1","Greschonig",Die,9:00,90,"HS1Strudlhof">,
<"260259",1,"Rep. Analysis f Physik 1","Muthsam",Don,13:00,90,"HS1Strudlhof">,
<"260226",1,"LinAlg Physik","Schmitt",Don,10:30,75,"HS1Strudlhof">,
<"260226",2,"LinAlg Physik","Schmitt",Fre,10:30,75,"HS1Strudlhof">,
<"260227",1,"PS LinAlg Physik","Schmitt",Fre,8:00,90,"HS1Strudlhof">,
<"260227",2,"PS LinAlg Physik","Schlosser",Mon,9:00,90,"HS2Strudlhof">,
<"260227",3,"PS LinAlg Physik","Schlosser",Die,9:00,90,"HS2Strudlhof">,
<"260227",4,"PS LinAlg Physik","Ploss",Mit,9:00,90,"HS1Strudlhof">,
<"260227",5,"PS LinAlg Physik","Ploss",Don,9:00,90,"HS1Strudlhof">,
<"260016",1,"Rep LinAlg Physik","Schmitt",Die,12:00,90,"HS1Strudlhof">,
<"250108",1,"Algebra 2","Summerer",Don,10:00,90,"HS3">,
<"250109",1,"PS Algebra 2","Summerer",Fre,15:00,45,"C209">,
<"250150",1,"Grundb mathm Logik","Kellner",Mit,14:00,90,"D101">,
<"250151",1,"PS Grundb mathm Logik","Mildenberger",Don,16:30,45,"C207">,
<"250165",1,"Funktionalanalysis 1","Feichtinger",Mon,9:00,45,"D101">,
<"250165",2,"Funktionalanalysis 1","Feichtinger",Mit,9:00,45,"D101">,
<"250165",3,"Funktionalanalysis 1","Feichtinger",Don,9:00,45,"D101">,
<"250112",1,"PS Funktionalanalysis 1","Feichtinger",Die,9:00,45,"D103">,
<"250022",1,"PDE 1","Hoffmann-Ostenhof",Mon,10:00,45,"A101">,
<"250022",2,"PDE 1","Hoffmann-Ostenhof",Die,10:00,45,"A101">,
<"250022",3,"PDE 1","Hoffmann-Ostenhof",Mit,10:00,45,"A101">,
<"250120",1,"PS PDE 1","Hoffmann-Ostenhof",Mit,17:00,45,"C207">,
<"250030",1,"Geschichte d Mathem.","Schmitt",Fre,13:30,90,"C209">,
<"250164",1,"Gruppentheorie","Auinger",Mit,11:00,90,"C207">,
<"250164",2,"Gruppentheorie","Auinger",Fre,11:00,90,"C207">,
<"250166",1,"PS Gruppenth.","Auinger",Mon,13:30,90,"D107">,
<"250134",1,"Gew DGL 2","Schmeiser",Fre,9:00,90,"C207">,
<"250095",1,"PS Gew DGL 2","Fellner",Fre,12:00,90,"D107">,
<"250114",1,"Komplexe Analysis 2","Haslinger",Mit,11:00,45,"C209">,
<"250114",2,"Komplexe Analysis 2","Haslinger",Fre,11:00,45,"C209">,
<"250115",1,"PS Komplexe Analysis 2","Haslinger",Die,16:30,45,"D101">,
<"250126",1,"Numerische Math. 2","Schichl",Mon,12:00,45,"C209">,
<"250126",2,"Numerische Math. 2","Schichl",Die,12:00,90,"C209">,
<"250127",1,"PS Numerische Math. 2","Schichl",Fre,11:00,45,"A101">,
<"250038",1,"Optimierung Variat.","Neumaier",Mon,11:00,45,"A101">,
<"250038",2,"Optimierung Variat.","Neumaier",Die,11:00,45,"A101">,
<"250038",3,"Optimierung Variat.","Neumaier",Mit,11:00,45,"A101">,
<"250110",1,"Biomathematik","metaBuergerSigmund",Mon,13:00,90,"D103">,
<"250110",2,"Biomathematik","metaBuergerSigmund",Die,13:00,90,"D103">,
<"250111",1,"PS Biomathematik","Buerger",Mon,11:00,45,"A109">,
<":308",1,"Stoch Prozesse","Schmidt",Die,11:00,90,"D103">,
<":308",2,"Stoch Prozesse","Schmidt",Mit,11:00,90,"D103">,
<"250064",1,"DiffGeo 2","Kriegl",Mon,11:00,45,"C206">,
<"250064",2,"DiffGeo 2","Kriegl",Die,11:00,45,"C206">,
<"250064",3,"DiffGeo 2","Kriegl",Mit,11:00,45,"C206">,
```

```
<"250078",1,"PS DiffGeo 2","Kriegl",Fre,10:00,90,"C206">,
<"250118",1,"Hoeh Wahrscheinlkth","F.Hofbauer",Mon,13:00,90,"C206">,
<"250118",2,"Hoeh Wahrscheinlkth","F.Hofbauer",Die,13:00,90,"C206">,
<"250104",1,"Analytische ZT","Burde",Mon,12:00,90,"D107">,
<"250104",2,"Analytische ZT","Burde",Die,12:00,90,"D107">,
<"250124",1,"Alg Geometrie","Mahnkopf",Mit,12:30,90,"C207">,
<"250124",2,"Alg Geometrie","Mahnkopf",Don,12:30,90,"C207">,
<"250113",1,"Colombeau Alg","Grosser",Mit,12:00,90,"D107">,
<"250113",2,"Colombeau Alg","Grosser",Don,11:30,90,"D103">,
<"250014",1,"math Methoden d Datenanalyse","Neumaier",Mit,12:00,90,"C209">,
<"250014",2,"math Methoden d Datenanalyse","Neumaier",Fre,12:00,90,"C209">,
<":309",1,"Einf Spieltheorie","Sigmund",Fre,10:00,90,"D103">,
<"250168",1,"Populationsdyn v Infektionskrankh","Capasso",Die,17:00,90,"C207">,
<"250141",1,"AK ZT","Rindler",Mon,8:00,45,"C209">,
<"250141",2,"AK ZT","Rindler",Die,8:00,45,"C209">,
<"250141",3,"AK ZT","Rindler",Mit,8:00,45,"C209">,
<"250141",4,"AK ZT","Rindler",Don,8:00,45,"C209">,
<"250100",1,"PS AK ZT","Eisenkoelbl",Mon,11:00,45,"HS3">,
<"250075",1,"AK Kombinatorik","Cigler",Die,14:00,90,"C207">,
<"250123",1,"AK PDE freie RWP","Matevosyan",Don,13:00,90,"C714">,
<"250137",1,"AK Moderne Analysis","Losert",Mon,13:00,90,"A101">,
<"250137",2,"AK Moderne Analysis","Losert",Die,13:00,90,"A101">,
<"250167",1,"Dispersion Attractors .. KleinGordon","Komech",Mon,11:00,90,"C714">,
<"250167",2,"Dispersion Attractors .. KleinGordon","Komech",Die,11:00,90,"C714">,
<"250099",1,"AK DiffGeo Matrix+classGroups","Neretin",Mon,12:00,90,"A109">,
<"250099",2,"AK DiffGeo Matrix+classGroups","Neretin",Die,12:00,90,"A109">,
<"250125",1,"AK DiffGeo sympl Geo","Michor",Don,11:30,90,"D101">,
<"250125",2,"AK DiffGeo sympl Geo","Michor",Fre,11:30,90,"D101">,
<"250105",2,"AK Topologie DiffTopo","Haller",Mit,12:00,90,"C206">,
<"250117",2,"AK WahrscheinlchkTH","F.Hofbauer",Fre,10:00,90,"D107">,
<"250027",1,"KO mathem 'Kolloquium","Rindler",Mon,17:00,180,"D103">,
<"250066",1,"SE Zahlentheorie","metaBaxaSchoissengeier",Don,15:00,90,"D101">,
<"250131",1,"SE Kombinatorik","Krattenthaler",Mon,13:00,90,"C714">,
<"250019",1,"SE DiffGleichungen","metaMarkowichMauser",Die,15:00,90,"C714">,
<"250020",1,"SE Angew Nichtlin Analysis","metaMarkowichMauser",Don,17:30,90,"C714">,
<"250116",1,"SE komplexe Analysis","Haslinger",Mon,14:30,90,"A101">,
<"250121",1,"SE Pseudodiff Operatoren","Hoermann",Mon,14:30,90,"D103">,
<"250122",1,"SE nonlinear distributional gemoetry","Kunzinger",Die,14:30,90,"D101">,
<"250135",1,"SE pde in der Zellbiologie","Schmeiser",Fre,11:30,90,"C206">,
<"250138",1,"SE harmonische Analyse","Losert",Mon,15:00,90,"D101">,
<"250145",1,"SE nonstandard Analysis","Hejtmanek",Mon,14:30,90,"Schmetterer">,
<"250171",1,"SE moderne Analysis","metaFeichtingerGroechenig",Mit,15:30,90,
                                                              "Schmetterer">,
<"250021",1,"SE mathematik und physik","Neumaier",Mon,13:30,90,"A109">,
<"250128",1,"SE Opti","metaNeumaierSchichl",Fre,9:00,90,"A101">,
<"250034",1,"SE mathem Populationsgenetik","Buerger",Don,11:30,90,"A101">,
<"250170",1,"SE biomathem Modelle","Capasso",Mit,11:00,90,"C714">,
<"250169",1,"Geschlechtsspezif Prob MathUnterr","Tanzberger",Mon,17:00,90,"D107">,
<"250101",1,"Einf mathem Arbeiten","Kunzinger",Mon,8:00,90,"HS1">,
<"250129",1,"Ang Mathem f LAK","Humenberger",Mon,10:00,90,"HS1">,
<"250129",2,"Ang Mathem f LAK","Humenberger",Don,10:00,45,"HS1">,
<"250106",1,"PS Ang Mathem f LAK","Humenberger",Mon,9:00,45,"C207">,
<"250106",2,"PS Ang Mathem f LAK","Dorfmayr",Don,9:00,45,"C207">,
<"250098",1,"Hoehere Analysis f LAK","Goetz",Die,9:00,90,"HS2">,
<"250098",2,"Hoehere Analysis f LAK","Goetz",Mit,9:00,90,"HS2">,
<"250159",1,"PS Hoehere Analysis f LAK","Goetz",Don,11:00,90,"HS1">,
<"250158",1,"SE LAK Algebra","Mitsch",Mon,13:30,90,"C207">,
<"250107",1,"SE LAK WKTh","Raith",Mit,12:00,90,"A101">,
```

```
<"250097",1,"Einf Fachdidaktik","Goetz",Die,11:30,90,"D101">,
<"250157",1,"SE Schulpraktikum","Koth",Die,17:00,90,"C209">,
<"250050",1,"SE Unterrichtsplanung","Malle",Don,17:00,90,"C209">,
<"250051",1,"SE Fachdidaktik","Malle",Mon,18:30,90,"C209">,
<"250048",1,"SE Didaktik d Mathem","Humenberger",Don,12:30,90,"C209">,
<"250163",1,"SE neue Entwickl d Fachdidaktik","Ulovec",Fre,13:00,90,"D101">,
<"250130",1,"Schulmathe 1 Zahlbereiche","Humenberger",Die,13:00,90,"D101">,
<"250094",1,"PS Schulmathe 1","Hauer-Typpelt",Don,10:00,45,"C209">,
<"250143",1,"Schulmathe 4 Vektorrech","Koth",Die,13:30,90,"C209">,
<"250144",1,"PS Schulmathe 4 Vektorrech","Koth",Die,10:30,45,"C209">,
<"250144",2,"PS Schulmathe 4 Vektorrech","Koth",Mit,10:30,45,"D101">,
<"250147",1,"Schulmathem 6 DiffRech","Malle",Don,14:00,90,"C209">,
<"250148",1,"PS Schulmathem 6 DiffRech","Malle",Die,15:00,45,"C209">
}
WZeit[1] = 0
WZeit[2] = 1
WZeit[3] = 0
WZeit[4] = 0
WZeit[5] = 0
WZeit[6] = 1
WZeit[7] = 0
WZeit[8] = 1
WZeit[9] = 0
WZeit[10] = 0
WZeit[11] = 1
WZeit[12] = 1
WZeit[13] = 1
WZeit[14] = 0
WZeit[15] = 1
WZeit[16] = 0
WZeit[17] = 0
WZeit[18] = 0
WZeit[19] = 0
WZeit[20] = 1
WZeit[21] = 0
WZeit[22] = 0
WZeit[23] = 0

WOrt[1] = 0
WOrt[2] = 0
WOrt[3] = 0
WOrt[4] = 1
WOrt[5] = 1
WOrt[6] = 0
WOrt[7] = 1
WOrt[8] = 0
WOrt[9] = 0
WOrt[10] = 0
WOrt[11] = 1
WOrt[12] = 1
WOrt[13] = 0
WOrt[14] = 0
WOrt[15] = 0
WOrt[16] = 0
```

Literaturverzeichnis

[Abr91] D. Abramson. Constructing school timetables using simulated annealing: sequential and parallel algorithms. Man. Sci., 37:98 – 113, 1991.

[ACG+99] G. Ausiello, P. Crescenzi, G. Gambosi, V. Kann, A. Marchetti-Spaccamela, and M. Protasi. Complexity and Approximation: Combinatorial optimization problems and their approximability properties. Springer-Verlag, 1999.

[AK89] E.H.L. Aarts and J.H.M. Korst. Simulated Annealing and Boltzmann machines. Wiley, Chichester, 1989.

[Bar99] R. Barták. Constraint programming: In pursuit of the holy grail. In Week of Doctoral Students (WDS99), IV, pages 555–564, Prague, June 1999. MatFyz-Press.

[BC94] Christian Bessiere and Marie-Odile Cordier. Arc-consistency and arc-consitency again. In Manfred Meyer, editor, Proceedings ECAI'94 Workshop on Constraint Processing, Amsterdam, 1994.

[BFR92] C. Bessiere, E.C. Freuder, and J.-R. Régin. Using constraint metaknowledge to reduce arc consistency computation. Artificial Intelligence, 107:125 – 148, 1992.

[BL84] E. Bonomi and J-L. Lutton. The n-city travelling salesman problem: Statistical mechanics and the metrolopis algorithm. SIAM Review, 26:551 – 568, 1984.

[DS90] G. Dueck and T. Scheuer. Threshold accepting: A general purpose optimization algorithm appearing superior to simulated annealing. J. Comp. Phys., 90:161 – 175, 1990.

[FGK93] R. Fourer, D. Gay, and B.W. Kernighan. AMPL: A Modeling Language for Mathematical Programming. The Scientific Press, San Francisco, CA, 1993.

[Fre89] Eugene C. Freuder. Partial Constraint Satisfaction. In Proceedings of the Eleventh International Joint Conference on Artificial Intelligence, IJCAI-89, Detroit, Michigan, USA, pages 278–283, 1989.

[Glo86] Fred Glover. Future paths for integer programming and links to artificial intelligence. Comput. Oper. Res., 13(5):533–549, 1986.

[HDT92] P. Van Hentenryck, Y. Deville, and C.-M. Teng. A generic arc-consistency algorithm and its specializations. Artificial Intelligence, 57:291 – 321, 1992.

[ilo] ILOG Solver. http://www.ilog.com/.

[JL00] Narendra Jussien and Olivier Lhomme. Local search with constraint propagation and conflict-based heuristics. In Proceedings of the Seventh National Conference on Artificial Intelligence (AAAI 2000), pages 169–174, Austin, TX, USA, August 2000.

[KGV83] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi. Optimization by simulated annealing. Science, Number 4598, 13 May 1983, 220, 4598:671–680, 1983.

[Kum92] Vipin Kumar. Algorithms for constraint-satisfaction problems: A survey. AI Magazine, 13(1):32–44, 1992.

[Mac75] Alan K. Mackworth. Consistency in networks of relations. Technical Report TR-75-03, Vancouver, BC, Canada, Canada, 1975.

[MH86] R. Mohr and T.C. Henderson. Arc and path consistency revisited. Artificial Intelligence, 28:225–233, 1986.

[Per92] M. Perlin. Arc consistency for factorable relations. Artificial Intelligence, 53:329 – 342, 1992.

[PWB07] Eduardo Pinheiro, Wolf-Dietrich Weber, and Luiz André Barroso. Failure trends in a large disk drive population. In 5th USENIX Conference on File and Storage Technologies (FAST 2007), 2007. http://labs.google.com/papers/disk_failures.pdf.

[Ree95] Colin R. Reeves. Modern Heuristic Thechniques for Combinatorial Problems. McGraw-Hill Book Company Europe, 1995.

[Rut94] Zsofi Ruttkay. Fuzzy constraint satisfaction. In Proceedings 1st IEEE Conference on Evolutionary Computing, pages 542–547, Orlando, 1994.

[Scu] Marius Scurtescu. Nautilus Scripts for Subversion. http://marius.scurtescu.com/node/85.

[Smi95] Barbara M. Smith. A Tutorial on Constraint Programming. Technical Report 95.14, 1995.

[sof] GNOME – Desktopmanager für Linux. http://www.gnome.org/about/.

[SW99] K. Stergiou and T. Walsh. Encodings of Non-Binary Constraint Satisfaction Problems. In National Conference on Artifical Intelligence (AAAI-99), Orlando, Florida, 1999.

Lebenslauf

Harald Schilly

Geburtsdaten: 1981-01-29, Wien, Österreich
Staatsbürgerschaft: Österreich

Ausbildung

1986 – 1990	Volksschule, Wien
1990 – 1999	Realgymnasium GRG 23, Anton-Baumgartnerstraße, Wien
seit 1999	Diplomstudium Mathematik und Physik, Universität Wien

Berufsweg

seit 2002	Trainer bei Bildungsmanagement G.m.b.H.
ab 2002	Trainer bei agami systems integration
seit 2003	Tutor an der Universität Wien